CSE 205: Digital Logic Design

Logic Circuits

- Logic Circuits: Combinational and Sequential
- Combinational Circuits
  - A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.
- Sequential Circuits
  - A sequential circuits employ storage elements and logic gates.
  - The outputs are a function of the inputs and the state of the storage elements.
  - The state of the storage elements, in turn, is a function of the previous inputs (and the previous state).

Combinational Circuits

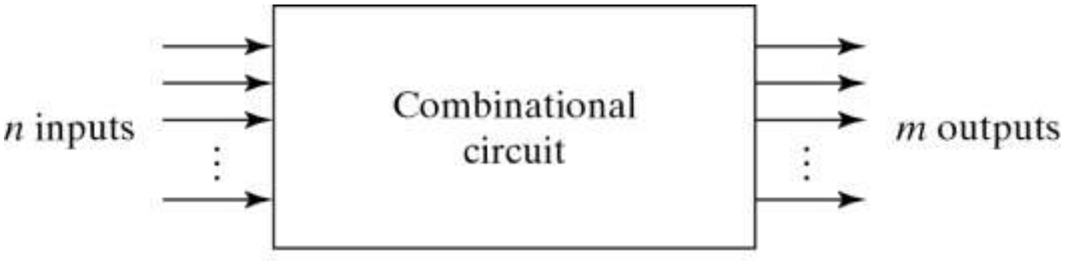
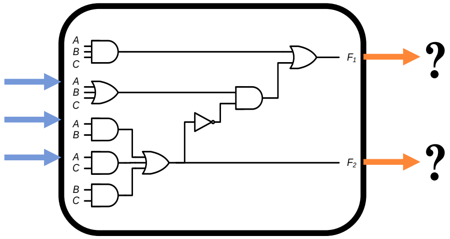


Fig. 4-1 Block Diagram of Combinational Circuit

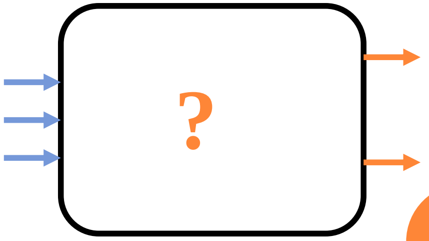
- The *n* input binary variables come from an external source.
- The *m* output variables are produced by the internal combinational logic circuit and go to an external destination.

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

$F_2 = AB + AC + BC$

$T_1 = A + B + C$

$T_2 = ABC$

$T_3 = F_2' \, T_1$

$F_1 = T_3 + T_2$

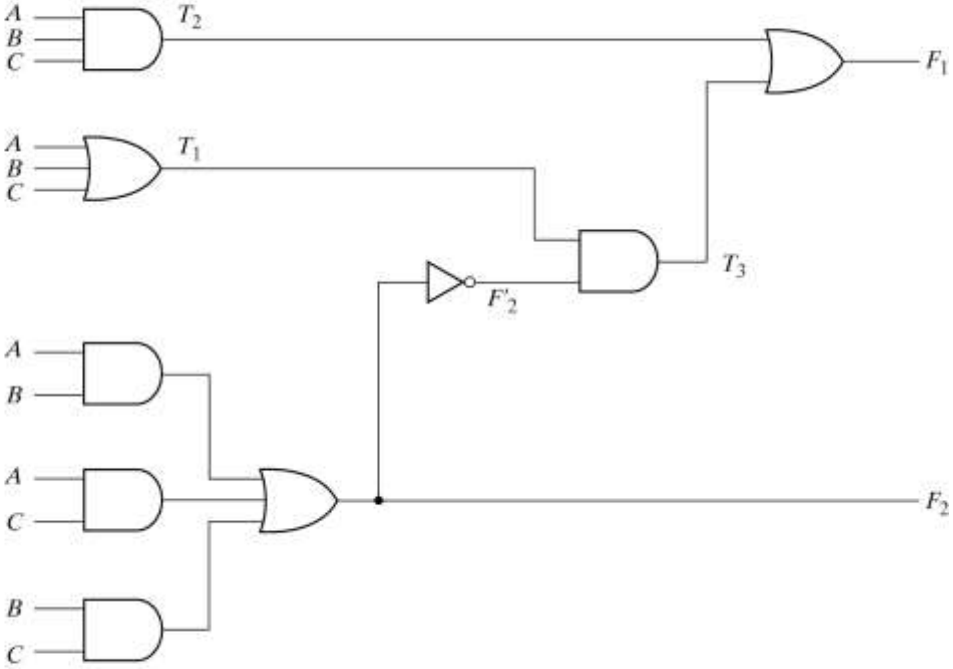


Fig. 4-2 Logic Diagram for Analysis Example

Or

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

Analysis Procedure
Truth Table Approach

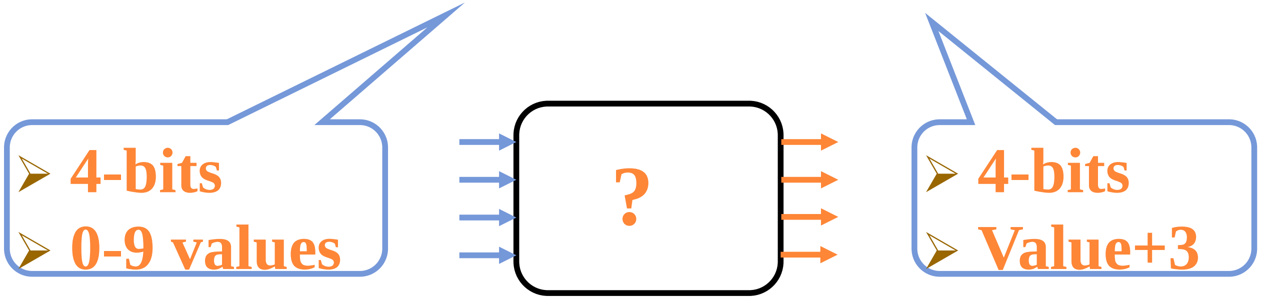
A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

DESIGN PROCEDURE

- 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 and verify it

Example:

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



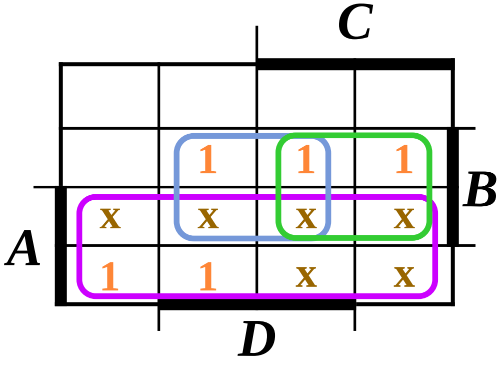
- **4-bits**
- **0-9 values**

?

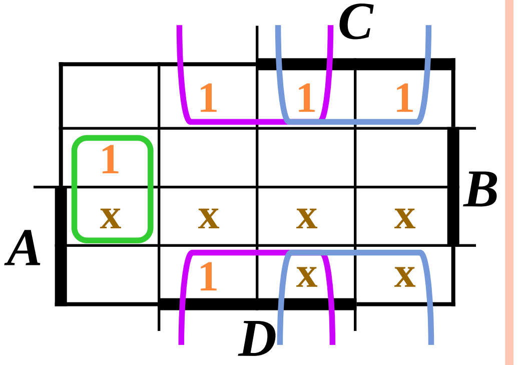
- **4-bits**
- **Value+3**

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



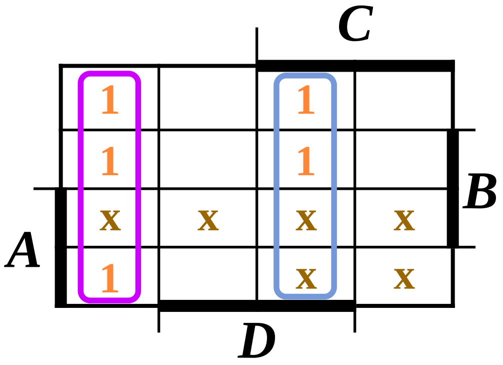
	1	1	1
x	x	x	x
1	1	x	x



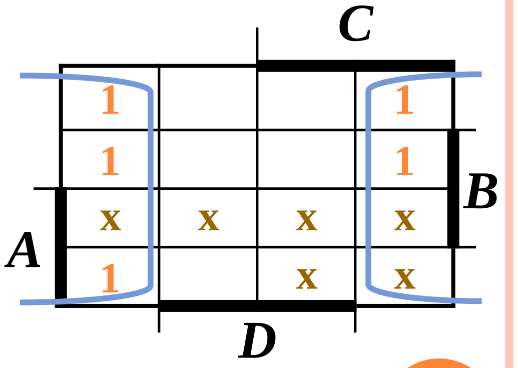
	1	1	1
1			
x	x	x	x
	1	x	x

w = *A*+*BC*+*BD*

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



1		1	
1		1	
x	x	x	x
1		x	x



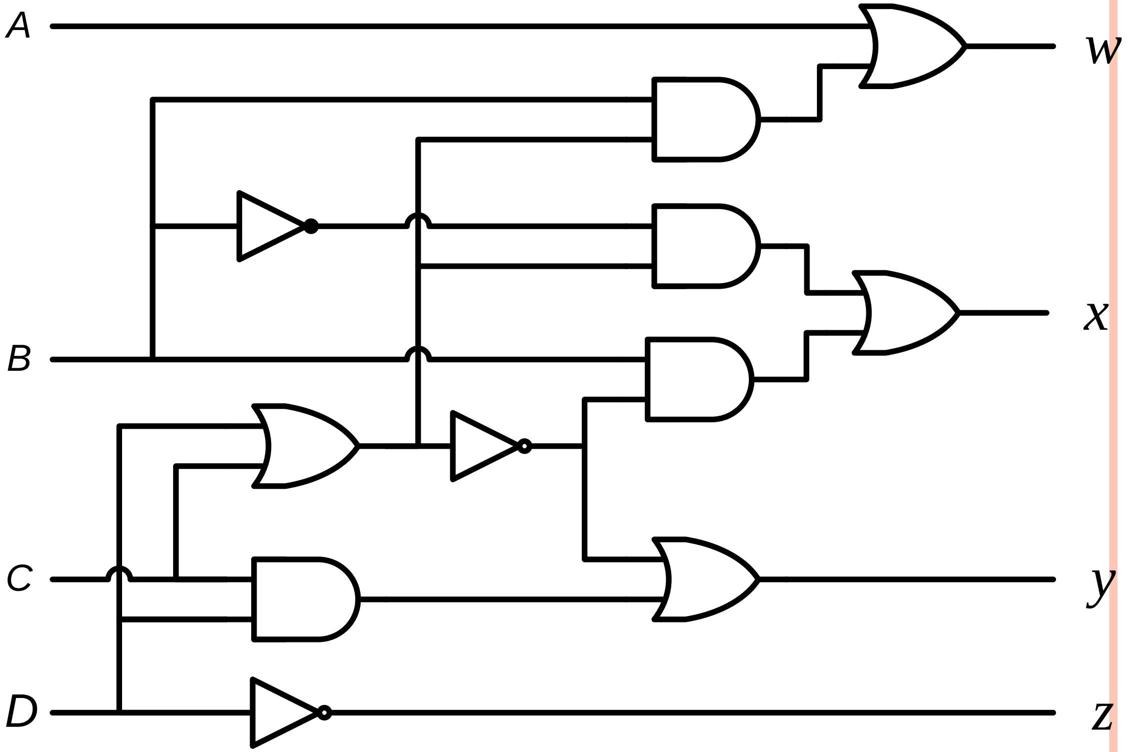
1			1
1			1
x	x	x	x
1		x	x

y = *C'D'*+*CD*

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



$w = A + B(C+D)$

$x = B'(C+D) + B(C+D)'$

$y = (C+D)' + CD$

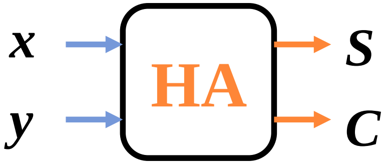
$z = D'$

Binary Adder

- The most basic arithmetic operation is the addition of two binary digits.
- A combinational circuit that performs the addition of two bits is **half adder**
- A adder performs the addition of 2 significant bits and a previous carry is called a **full adder**

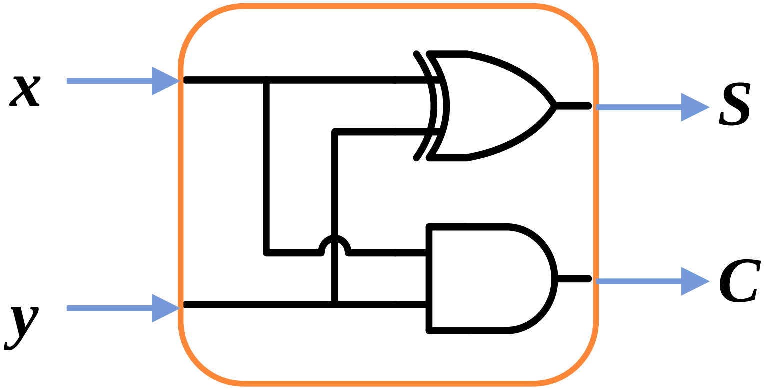
Binary Adder

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



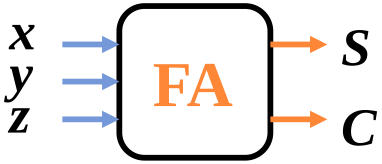
x y	*C S*
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

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



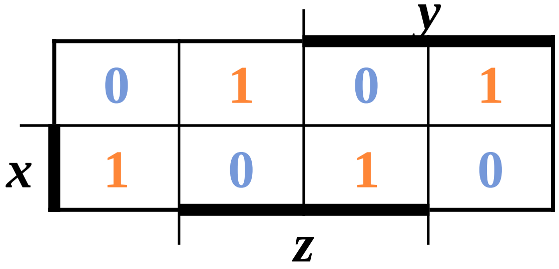
Binary Adder

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



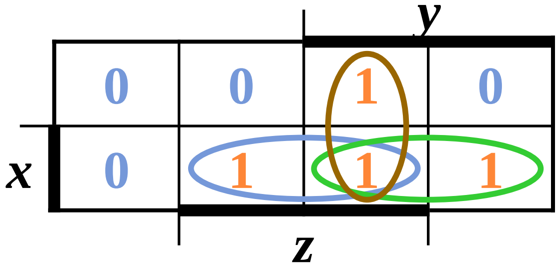
$$\begin{array}{cr} & x \\ + & y \\ + & z \\ \hline & C\ \ S \end{array}$$

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



0	1	0	1
1	0	1	0

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



0	0	1	0
0	1	1	1

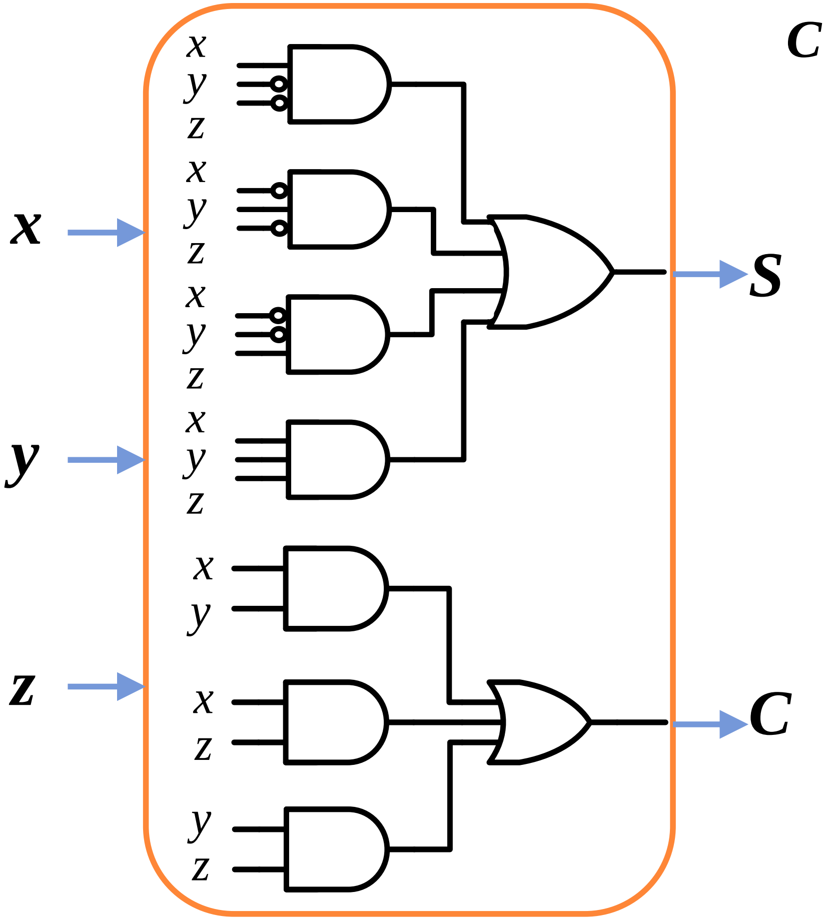
$C = xy + xz + yz$

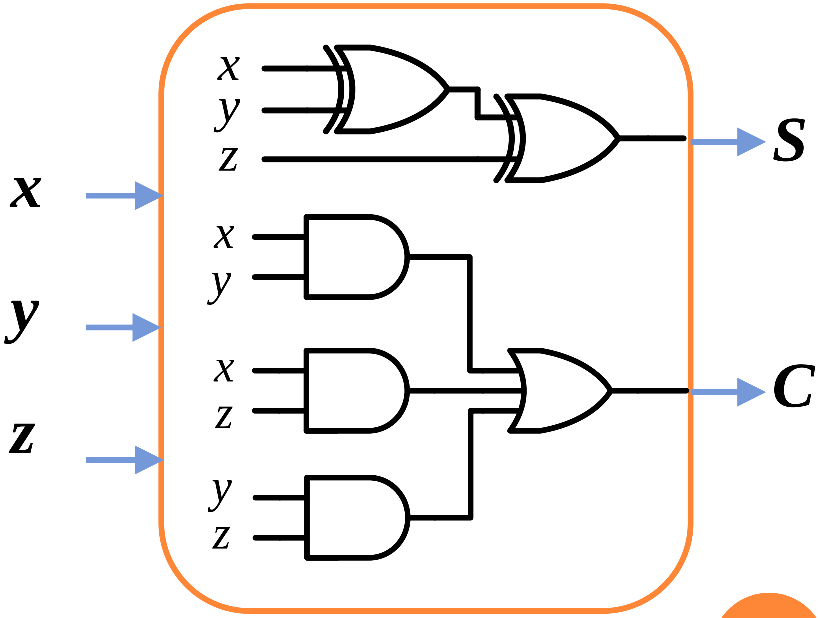
Binary Adder

- Full Adder

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

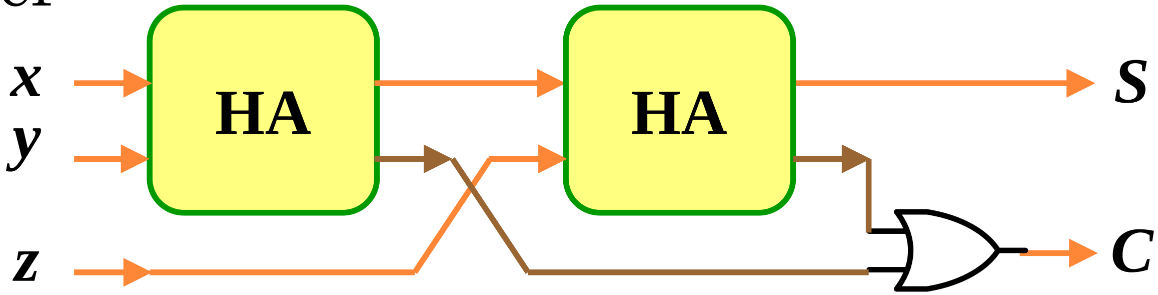
$C = xy + xz + yz$

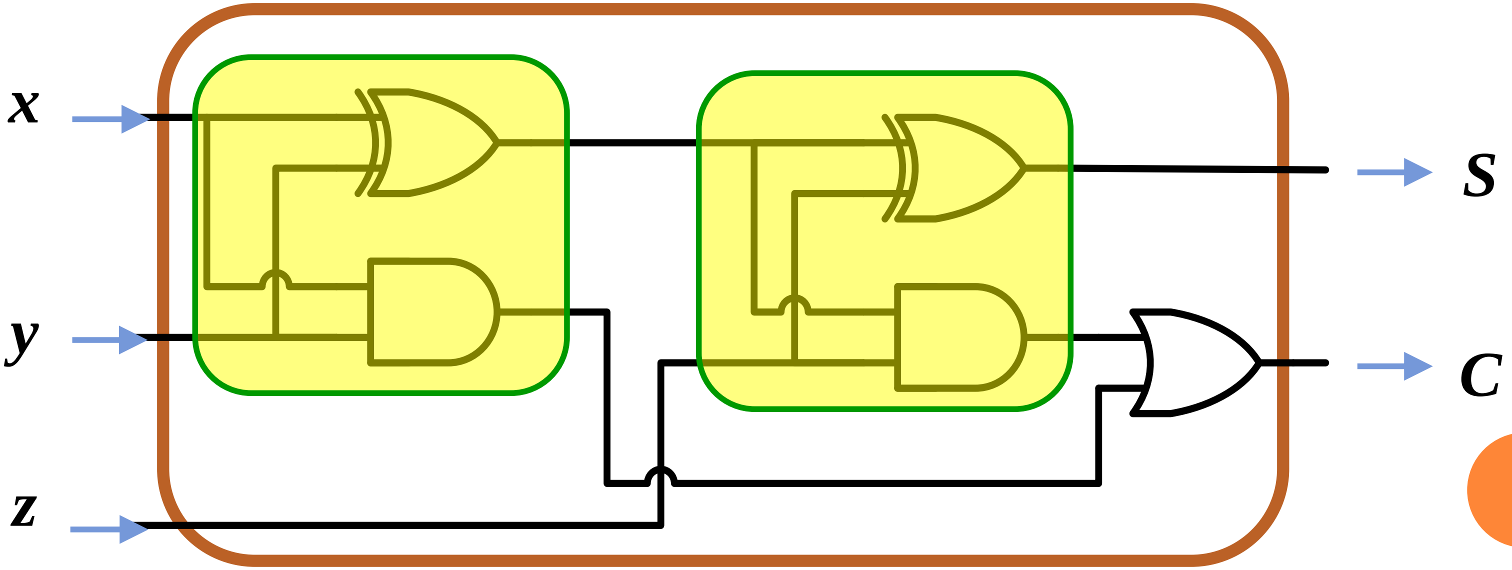




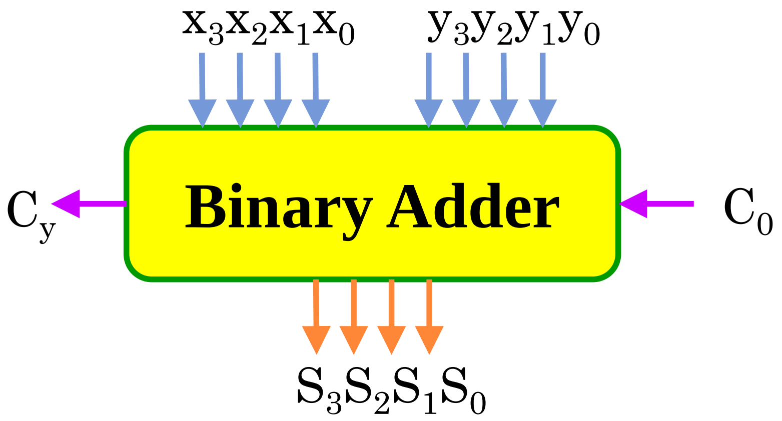
Binary Adder

- Full Adder



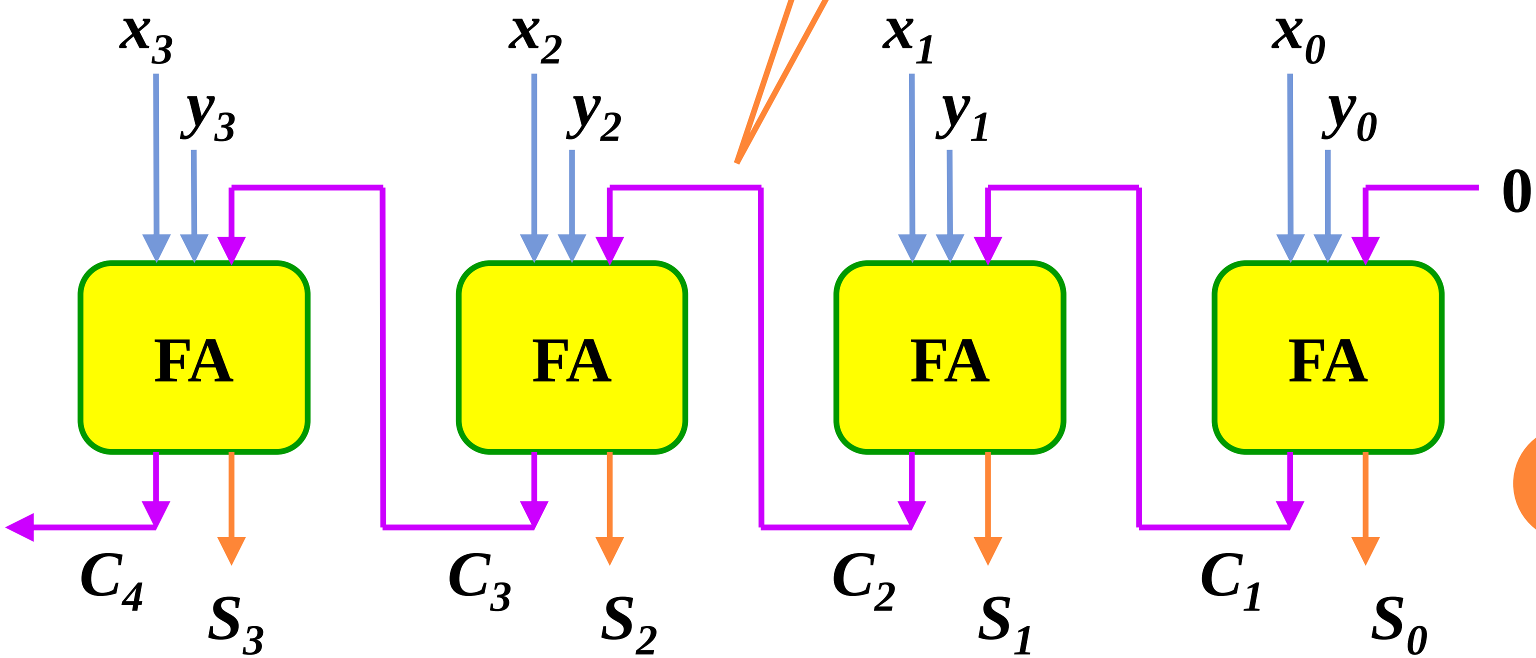


Binary Adder



Carry Propagate Addition

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



FOUR-BIT BINARY ADDER

- Carry bits must “ripple” through each stage of a multi-bit adder before the output settles down to the correct result.

- Significantly slower --> Rippling effect of carry

- For an *n* bit adder, Propagation delay = (Number of gate level x Average gate delay) x (number of bits)

Carry Lookahead Logic

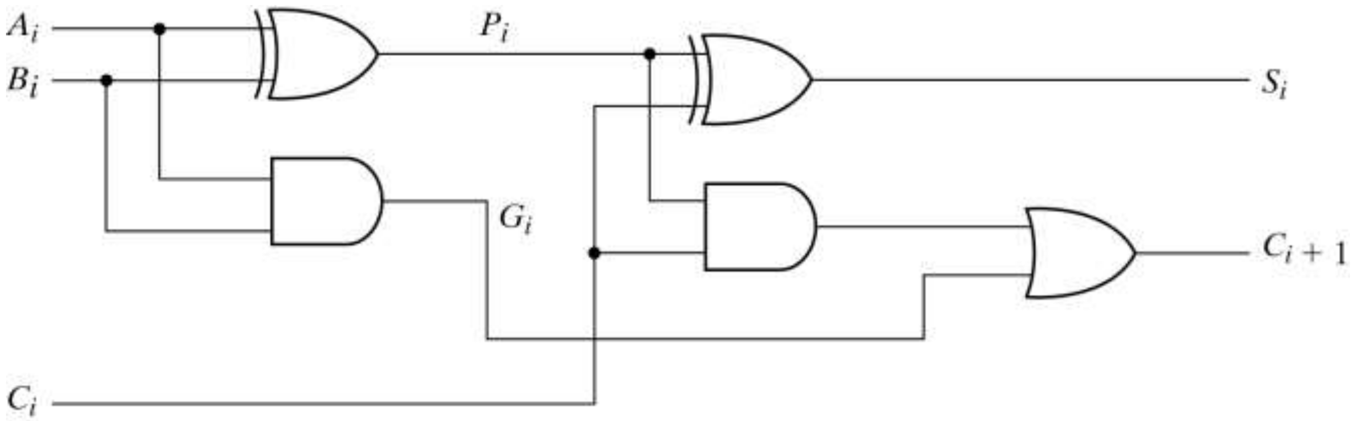


Fig. 4-10 Full Adder with P and G Shown

- Carry Propagate $P_i = A_i \oplus B_i$
- Carry Generate $G_i = A_i B_i$
- $S_i = P_i \oplus C_i$
- $C_{i+1} = G_i + C_i P_i$

Carry Lookahead Logic

- All carries can be generated simultaneously
  - $C_2 = G_1 + P_1C_1$
  - $C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1C_1$
  - $C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1C_1$

Carry Lookahead Logic

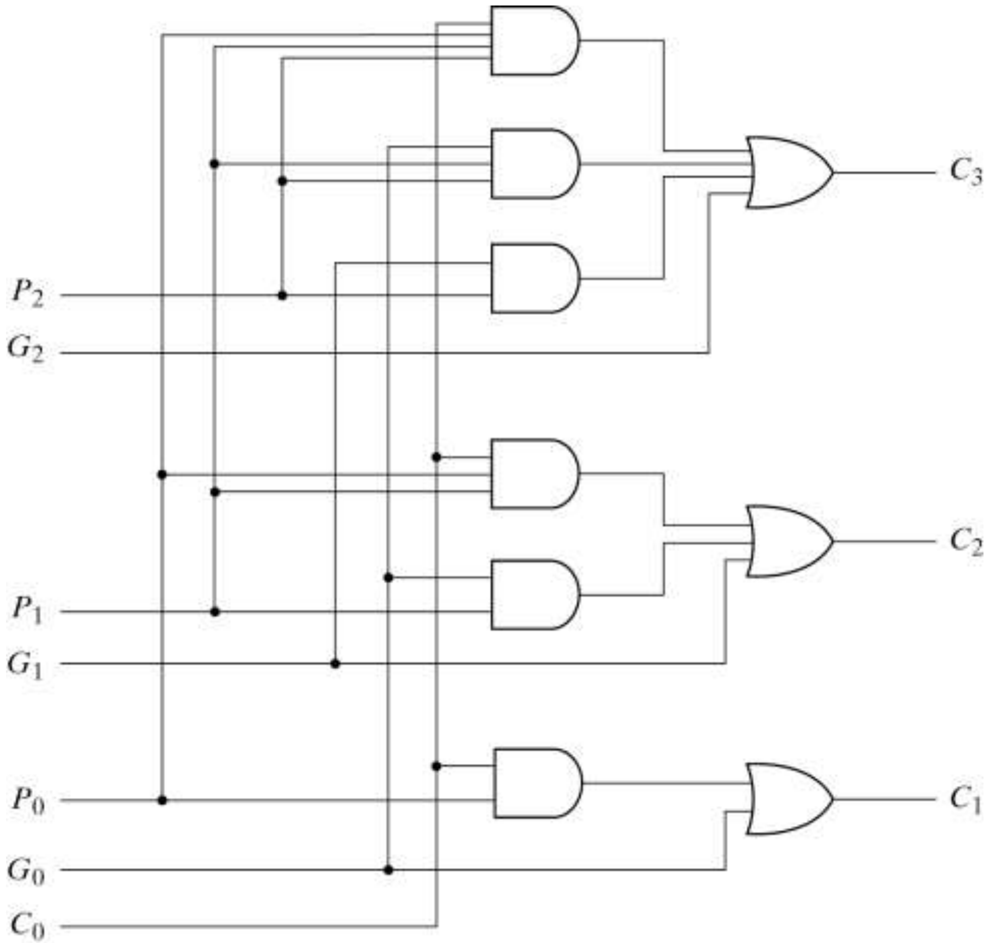


Fig. 4-11 Logic Diagram of Carry Lookahead Generator

4-Bit Adder with Carry Lookahead

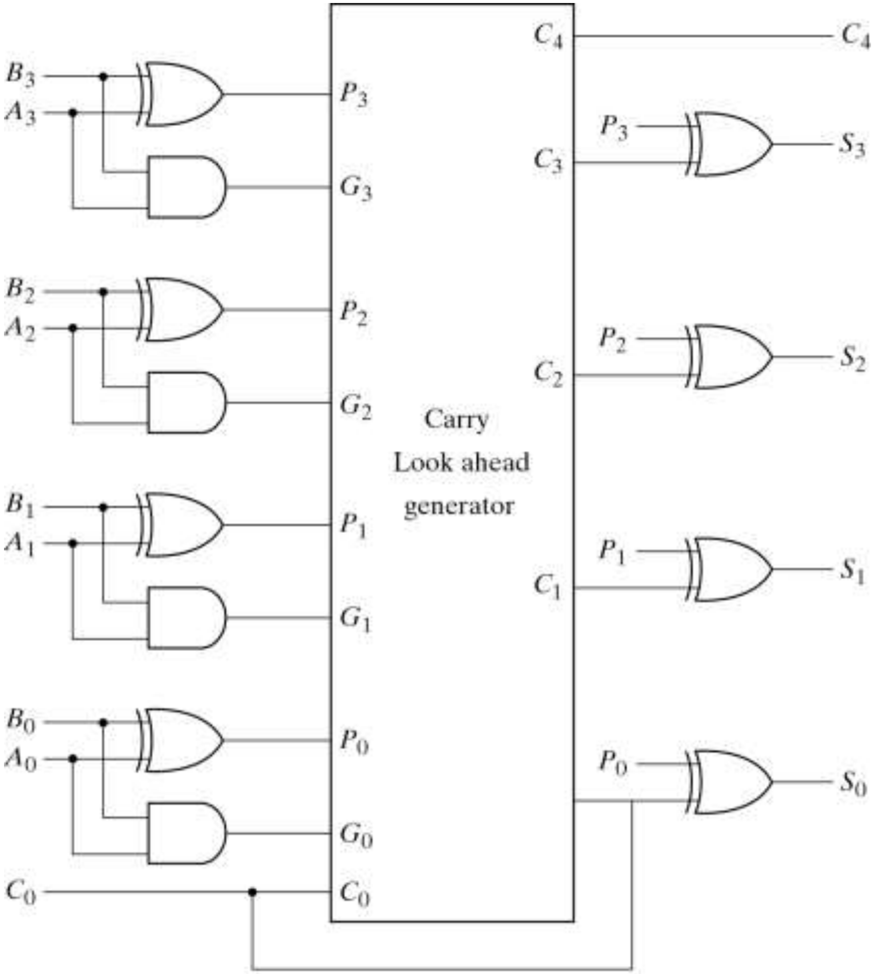
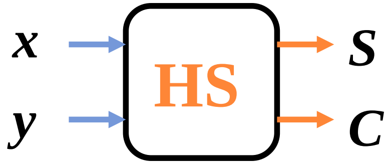


Fig. 4-12 4-Bit Adder with Carry Lookahead

Binary Subtractor

- Half Subtractor
  - Produces *x* -*y*
  - D – difference

x	*y*	*B*	*D*
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

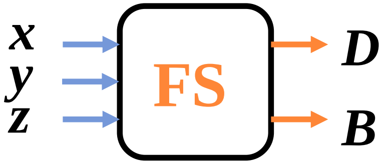


$$\begin{array}{r} x \\ -\ \ y \\ \hline B\ \ D \end{array}$$

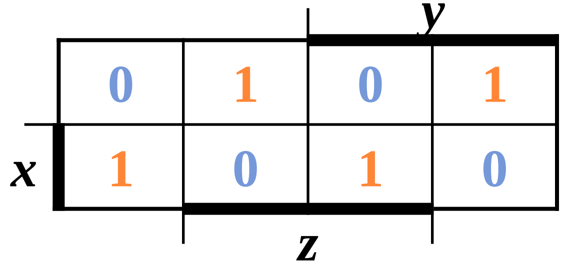
- D = *x'y* +*xy'* = *S* of half adder
- B = *x'y*

Binary Adder

- Full Subtractor
  - $(x - y) - z$; where z represents a borrow

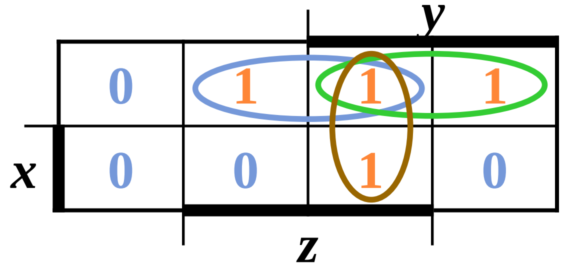


x y z	B D
0 0 0	0 0
0 0 1	1 1
0 1 0	1 1
0 1 1	1 0
1 0 0	0 1
1 0 1	0 0
1 1 0	0 0
1 1 1	1 1



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

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

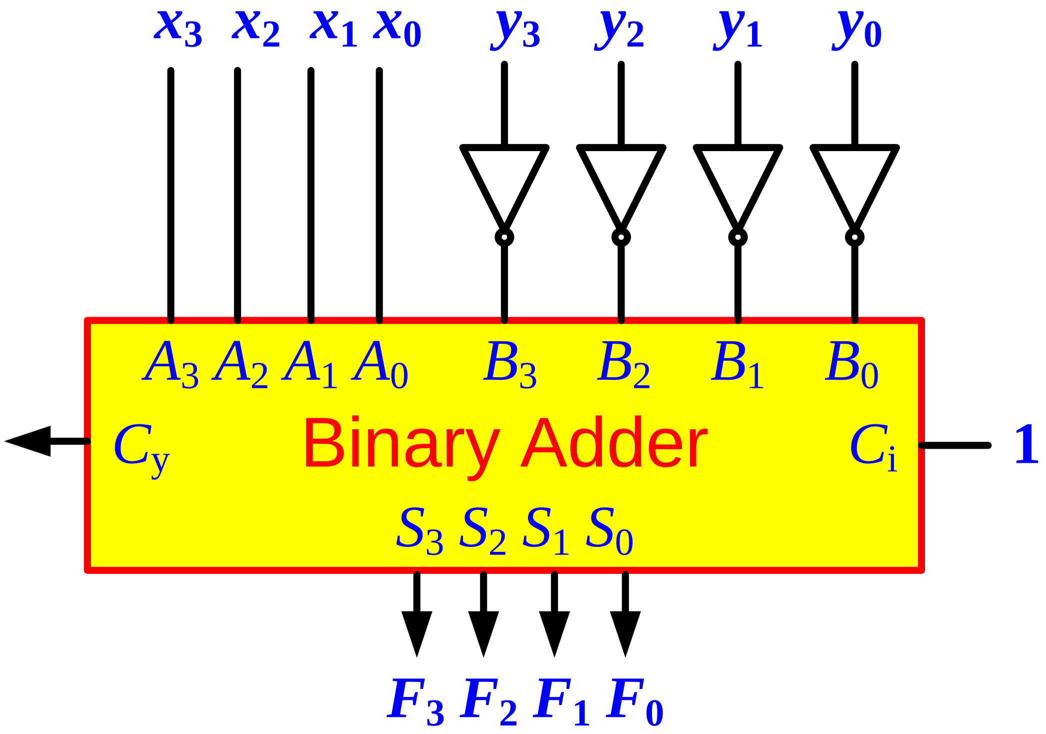


			y	y
	0	1	1	1
x	0	0	1	0
		z	z	

$$B = x'y + x'z + yz$$

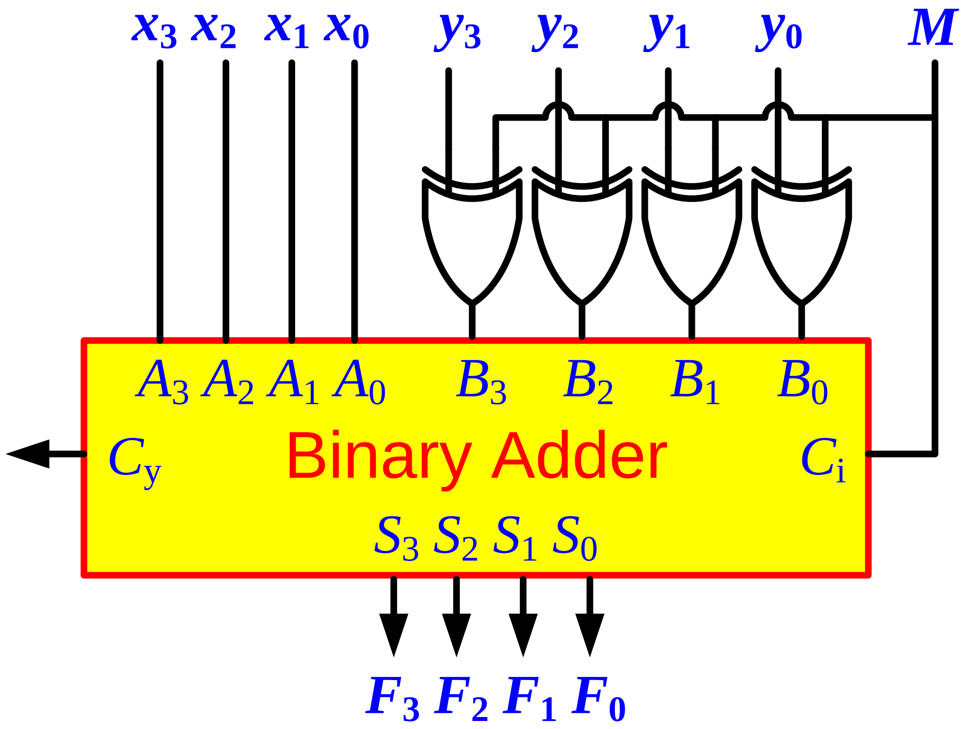
Binary Subtractor

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



Binary Adder/Subtractor

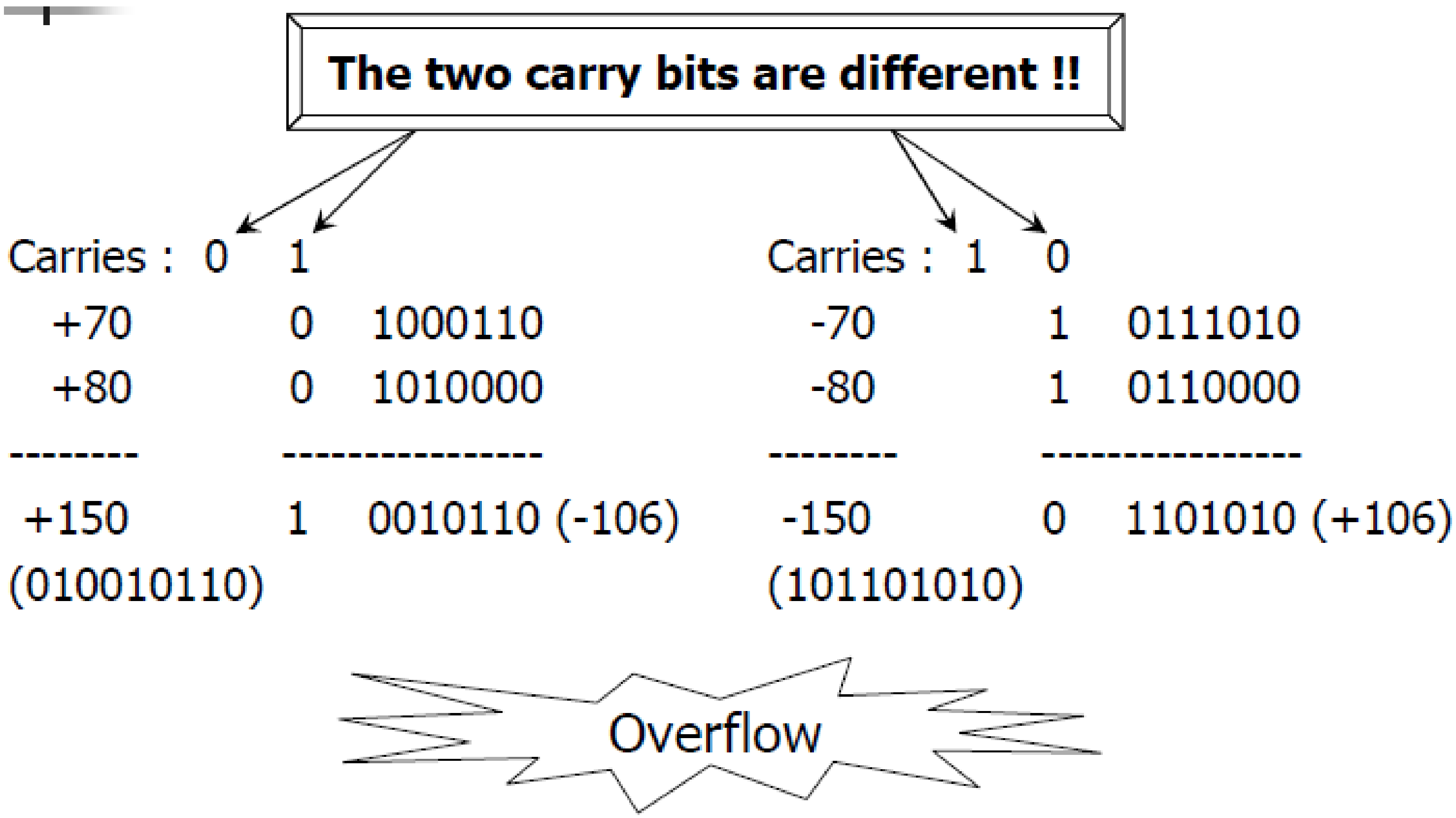
- M: Control Signal (Mode)
  - M=0 ➔ $F = x + y$
  - M=1 ➔ $F = x – y$



OVERFLOW

- An overflow occurs when two number of n digits each are added and the sum occupies n+1 digits
- When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant 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
  - Extra overflow detection circuits are required
- An overflow can only occur when two numbers added are **both positive or both negative**

Overflow



The two carry bits are different !!

Carries : 0 1

+70 0 1000110

+80 0 1010000

-------- ----------------

+150 1 0010110 (-106)

(010010110)

Carries : 1 0

-70 1 0111010

-80 1 0110000

-------- ----------------

-150 0 1101010 (+106)

(101101010)

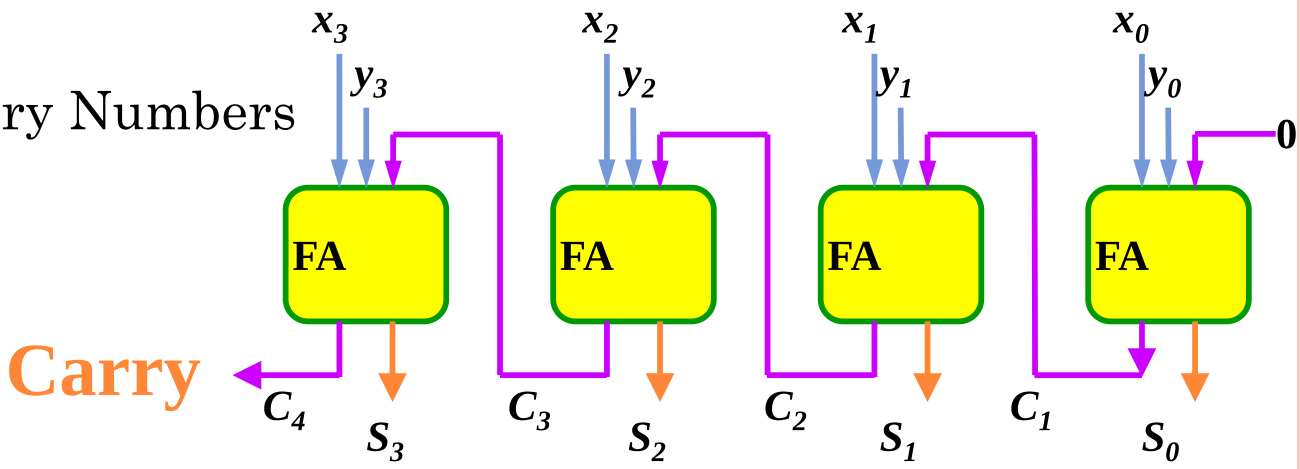
Overflow

Overflow

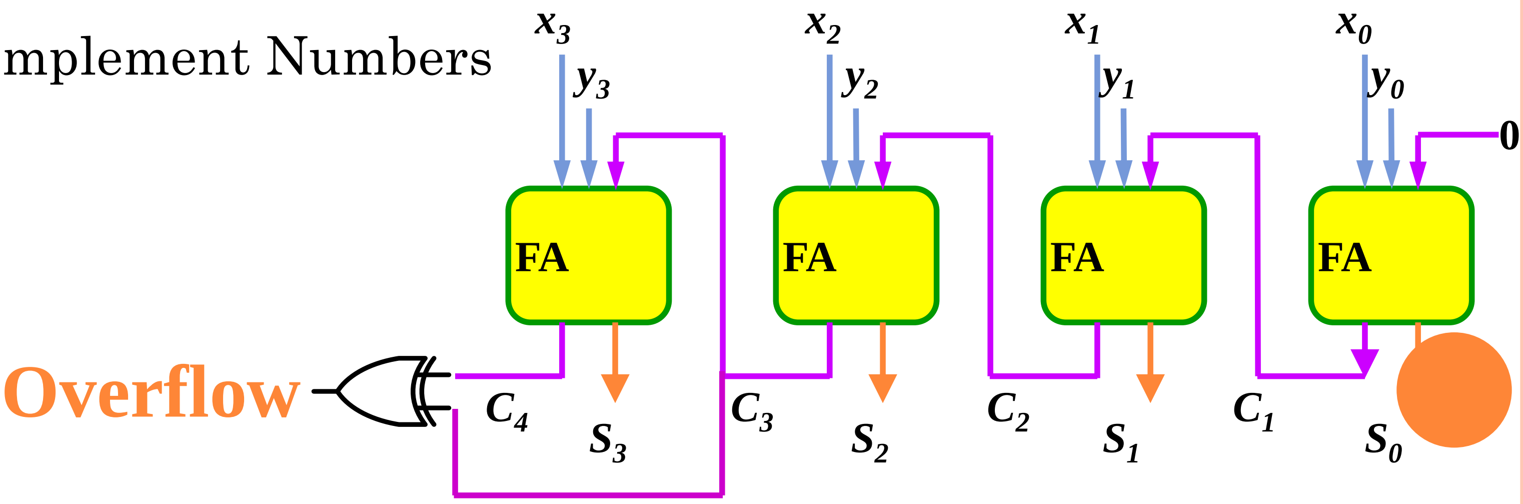
INPUTS			OUTPUTS		
A_{sign}	B_{sign}	CARRY IN	CARRY OUT	SUM_{sign}	OVERFLOW
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

Overflow

- Unsigned Binary Numbers



- 2's Complement Numbers



Decimal Adder

- A decimal adder requires a minimum of 9 inputs
- and 5 outputs
  - 1 digit requires 4-bit
  - Input: 2 digits + 1-bit carry
  - Output: 1 digit + 1-bit carry
- BCD adder
  - Perform the addition of two decimal digits in BCD, together with an input carry from a previous stage
  - The output sum cannot be greater than 19 (9+9+1)

BCD Adder

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

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

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	= 12	1	0 0 1 0

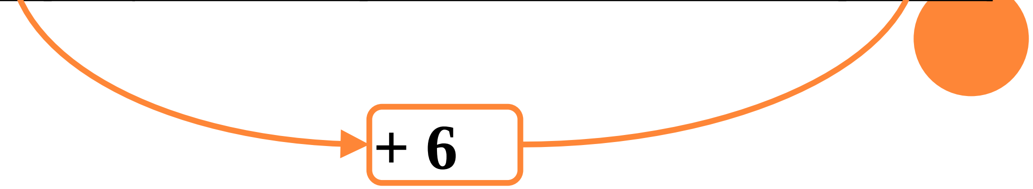
Invalid Code

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	✘

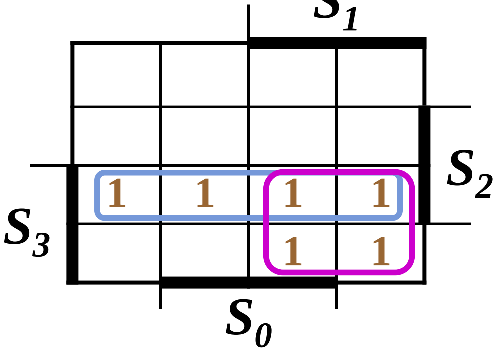


+ 6

BCD Adder

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

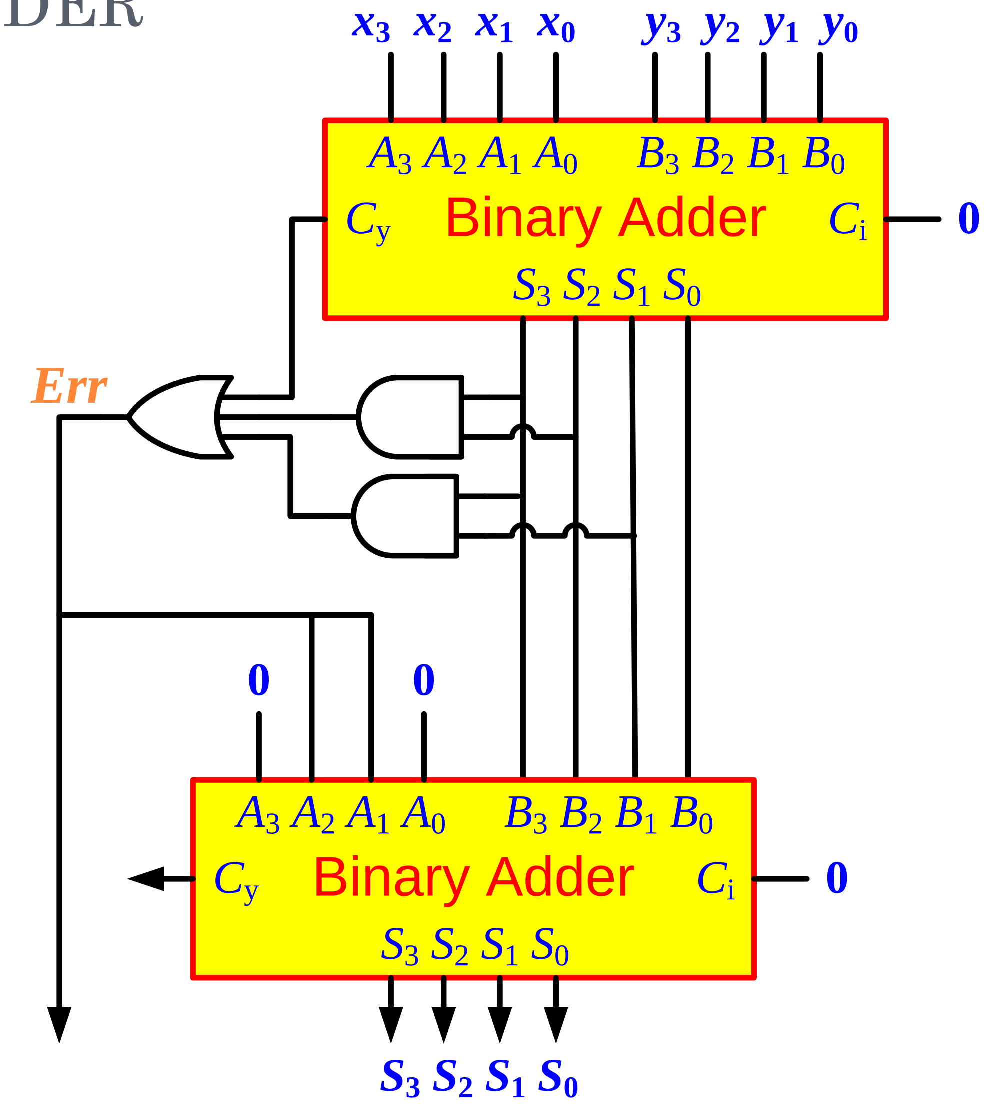
S_3 S_2 S_1 S_0	*Err*
0 0 0 0	0
1 0 0 0	0
1 0 0 1	0
1 0 1 0	1
1 0 1 1	1
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1



1	1	1	1
		1	1

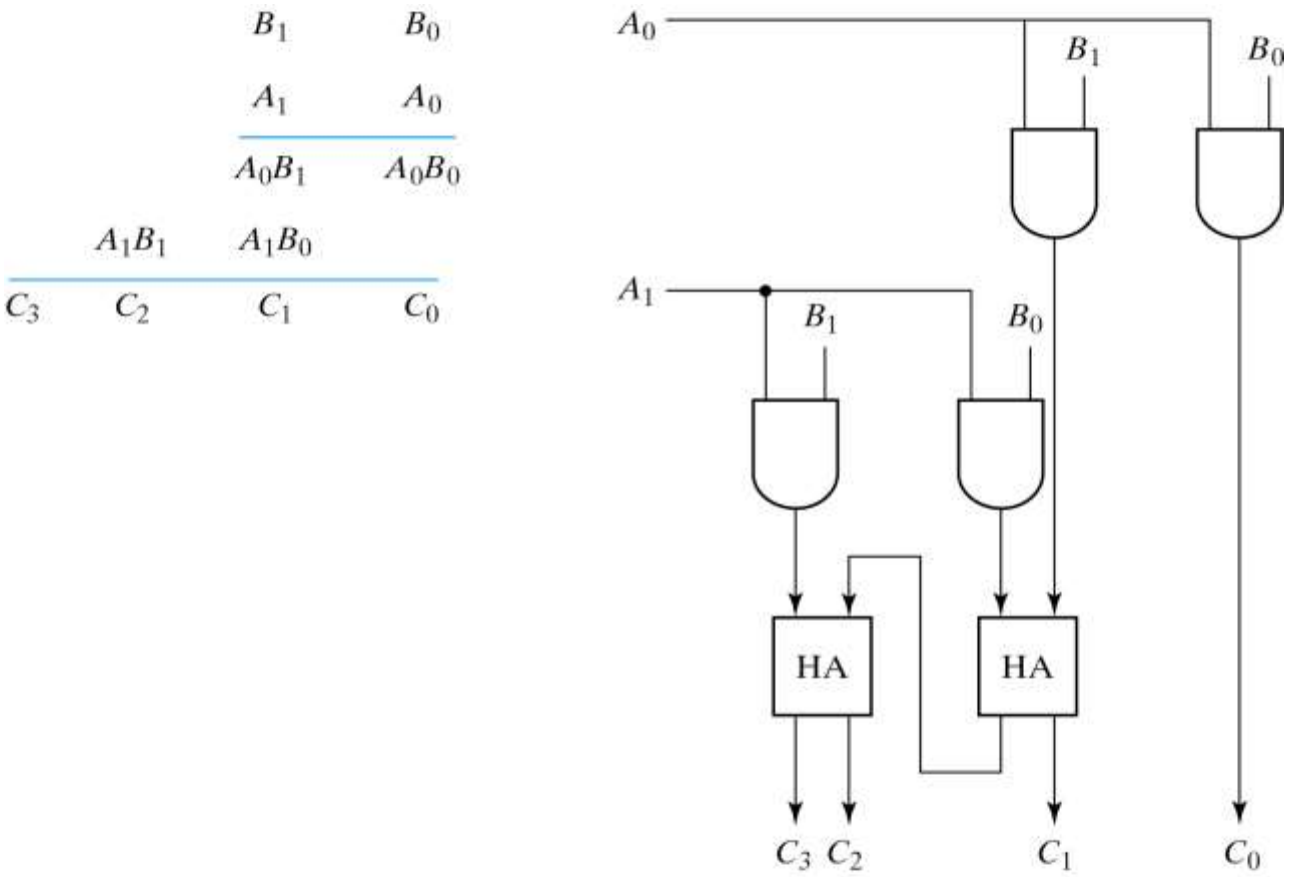
$$Err = S_3\,S_2 + S_3\,S_1$$

BCD Adder



x_3 x_2 x_1 x_0
y_3 y_2 y_1 y_0
A_3 A_2 A_1 A_0
B_3 B_2 B_1 B_0
C_y
Binary Adder
C_i
0
S_3 S_2 S_1 S_0
Err
0
0
A_3 A_2 A_1 A_0
B_3 B_2 B_1 B_0
C_y
Binary Adder
C_i
0
S_3 S_2 S_1 S_0
S_3 S_2 S_1 S_0

BINARY MULTIPILIER



$$\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}$$

Fig. 4-15 2-Bit by 2-Bit Binary Multiplier

Binary Multipilier

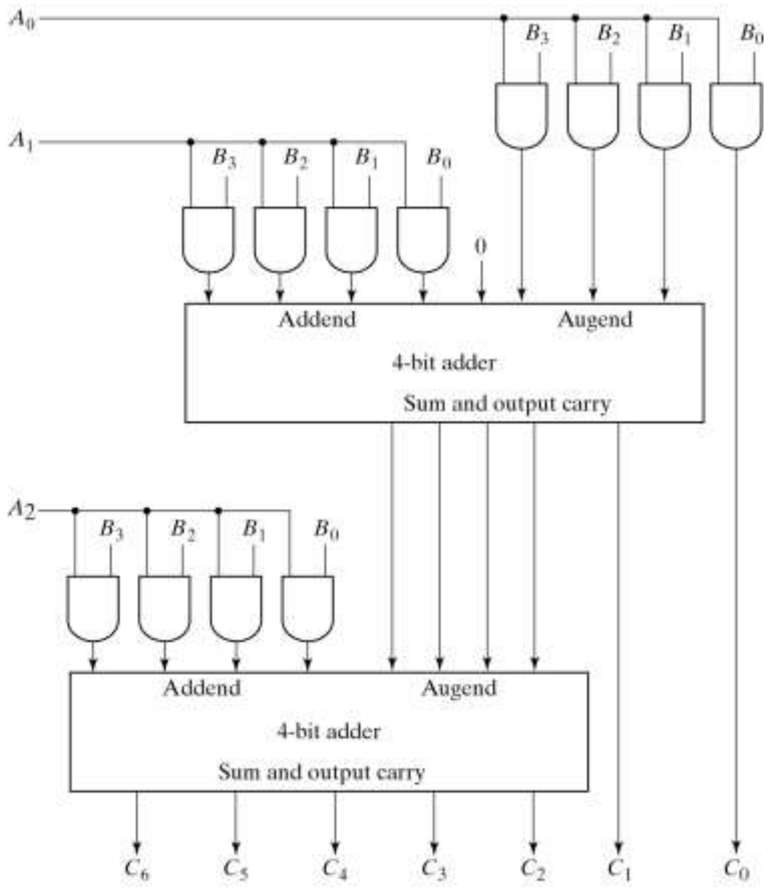
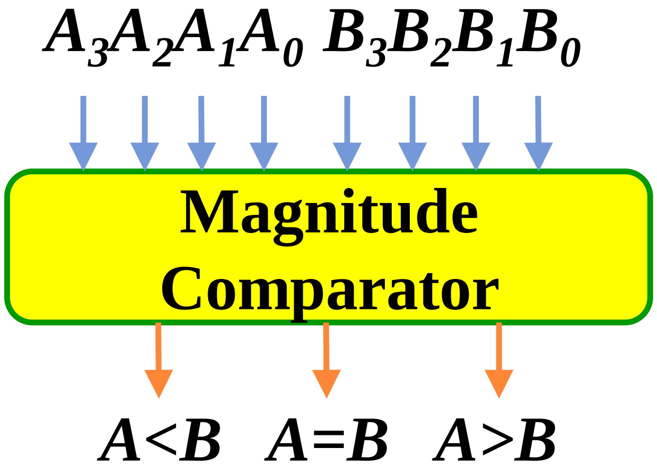


Fig. 4-16 4-Bit by 3-Bit Binary Multiplier

Magnitude Comparator

- Compare 4-bit number to 4-bit number
  - 3 Outputs: < , = , >
  - Expandable to more number of bits



$A_3A_2A_1A_0$ $B_3B_2B_1B_0$

Magnitude Comparator

A<B A=B A>B

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

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

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

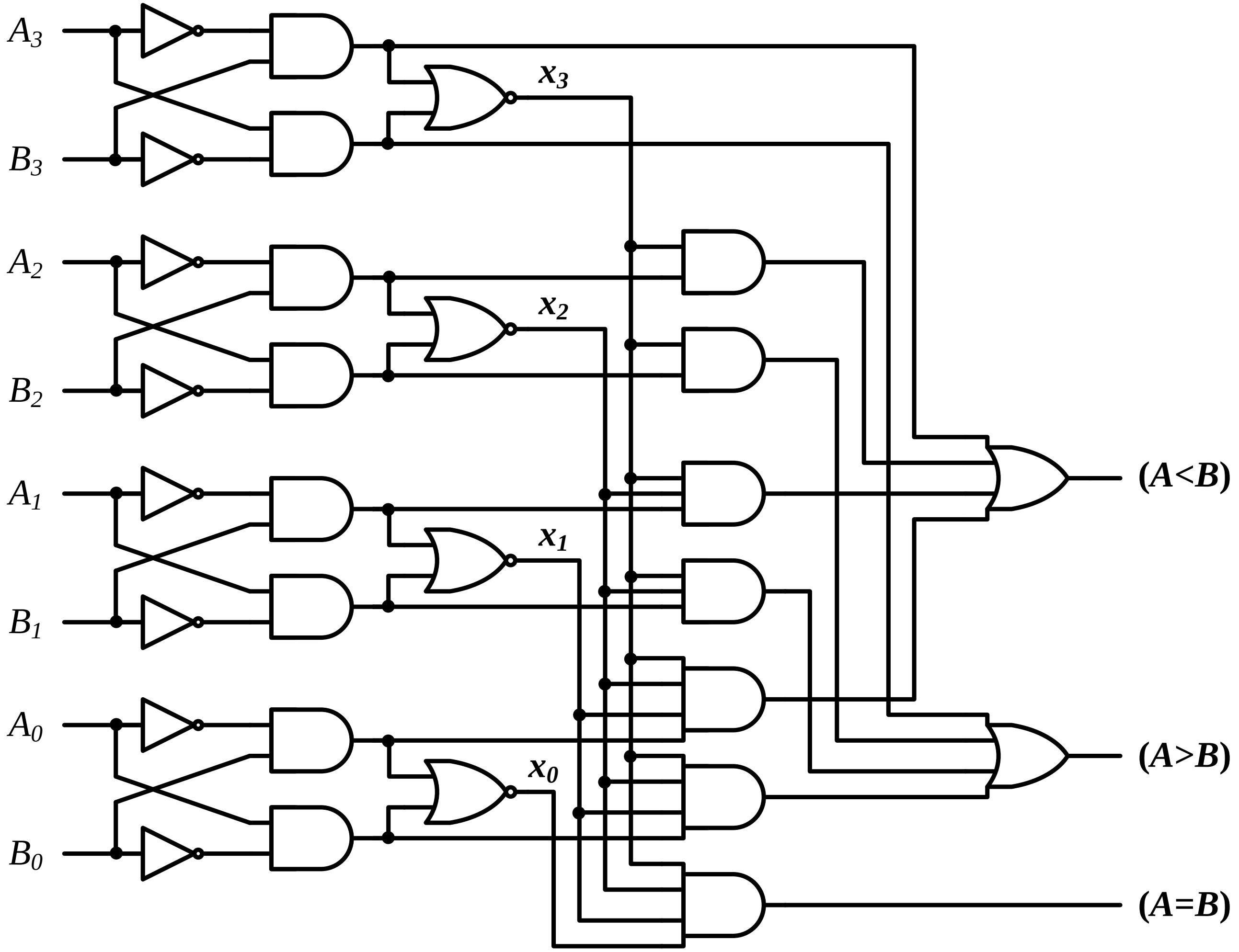
$$x_0 = \overline{A}_0\,\overline{B}_0 + A_0\,B_0$$

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

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

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

Magnitude Comparator

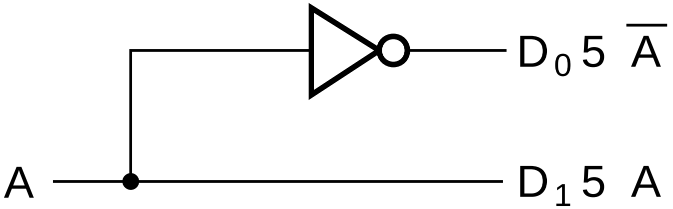


Decoders

- A decoder is a combinational circuit that converts binary information from *n input lines to an 2^n unique output lines.*
- 1-to-2-Line Decoder

A	D_0	D_1
0	1	0
1	0	1

(a)

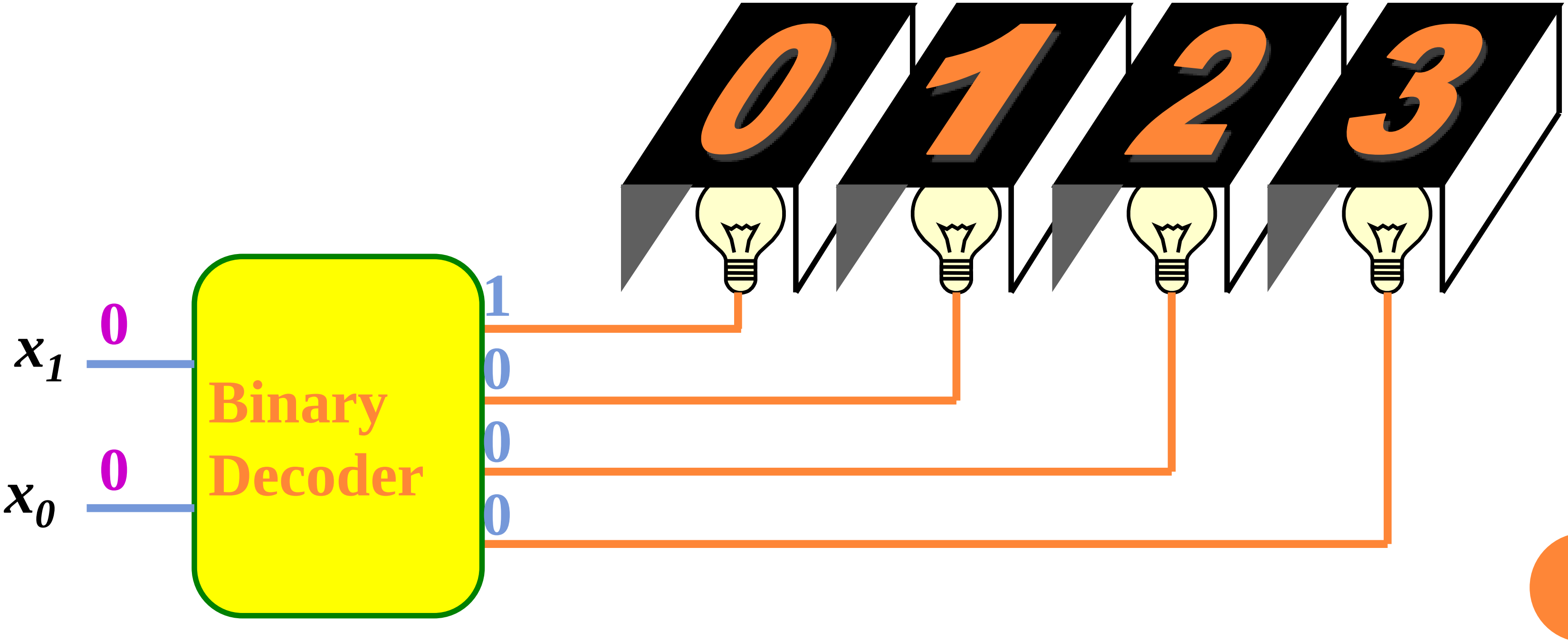


(b)

DECODERS

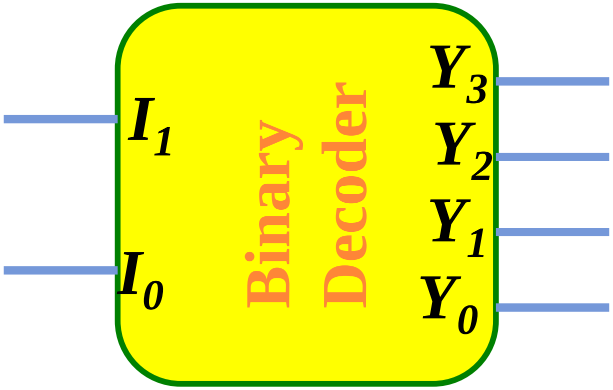
- Extract "*Information*" from the code
- Binary Decoder
  - Example: 2-bit Binary Number

Only *one* lamp will turn on

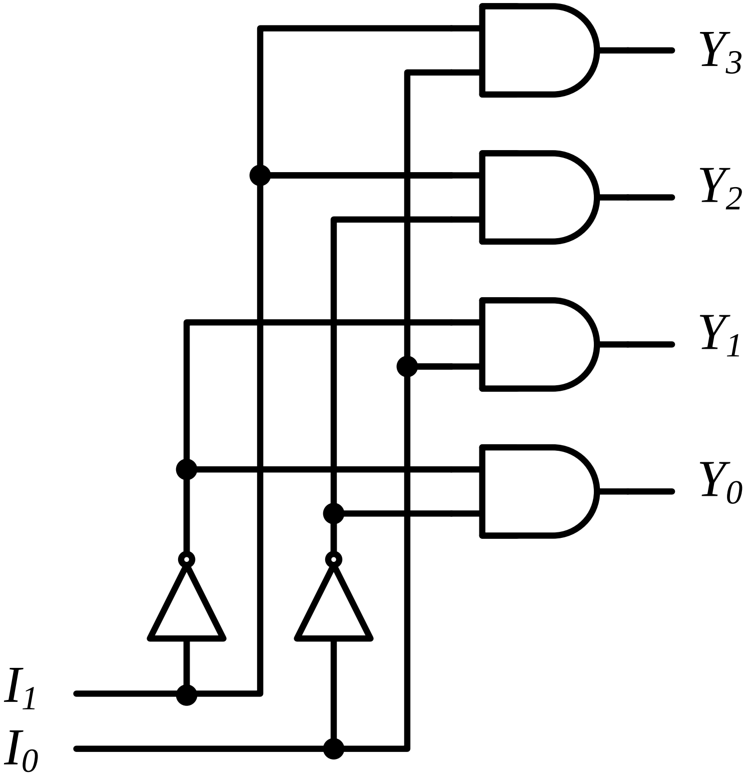


Decoders

- 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

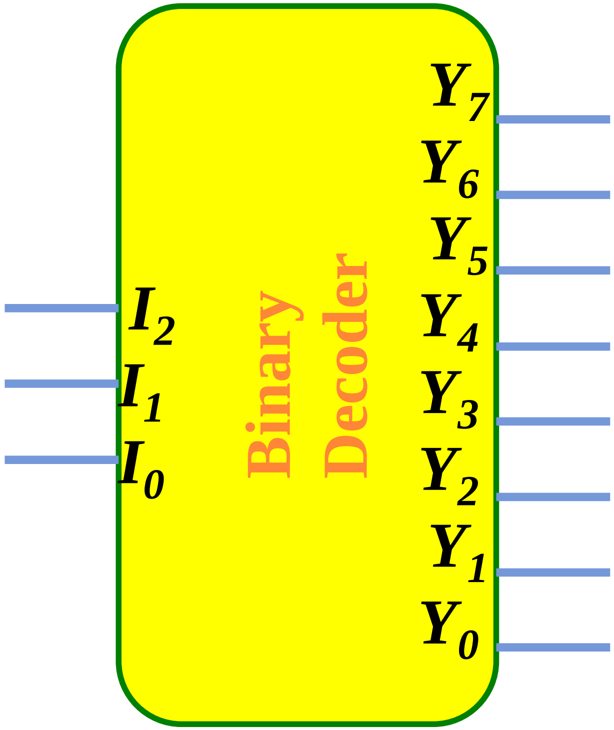


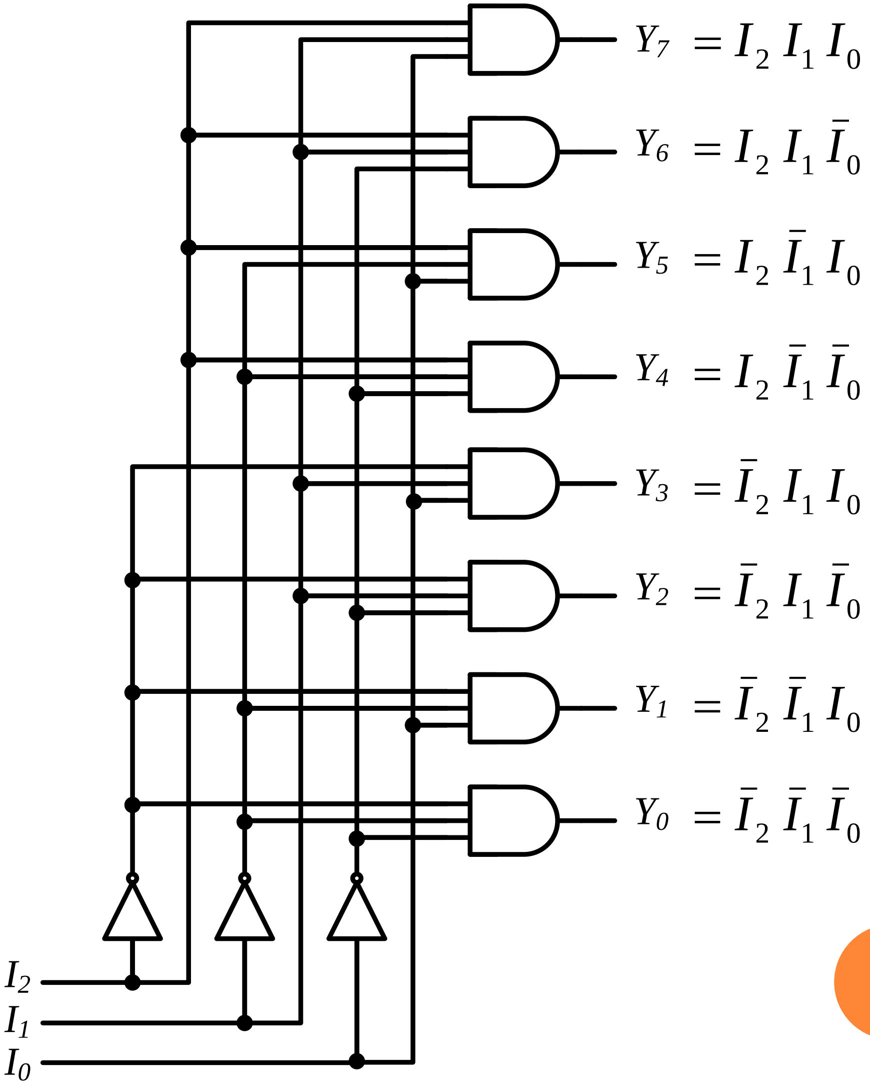
$$Y_3 = I_1\, I_0 \qquad Y_2 = I_1\, \bar{I}_0$$

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

DECODERS

- 3-to-8 Line Decoder

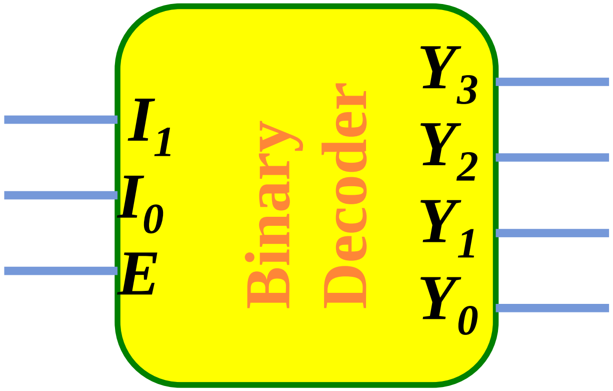




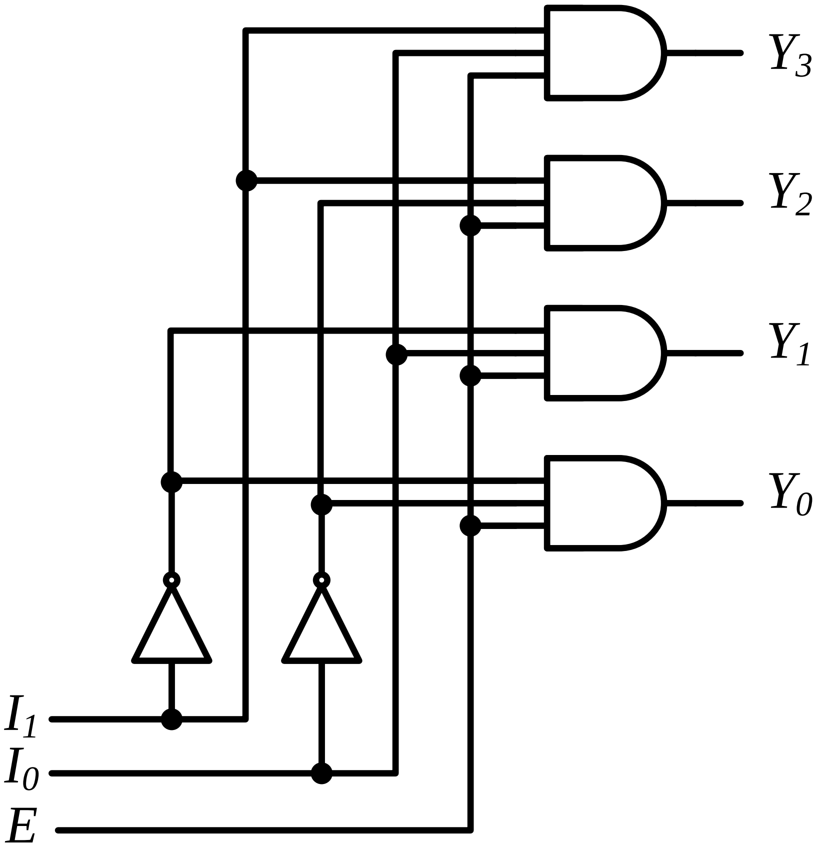
$Y_7 = I_2 I_1 I_0$
$Y_6 = I_2 I_1 \bar{I}_0$
$Y_5 = I_2 \bar{I}_1 I_0$
$Y_4 = I_2 \bar{I}_1 \bar{I}_0$
$Y_3 = \bar{I}_2 I_1 I_0$
$Y_2 = \bar{I}_2 I_1 \bar{I}_0$
$Y_1 = \bar{I}_2 \bar{I}_1 I_0$
$Y_0 = \bar{I}_2 \bar{I}_1 \bar{I}_0$
I_2
I_1
I_0

Decoders

- "*Enable*" Control



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

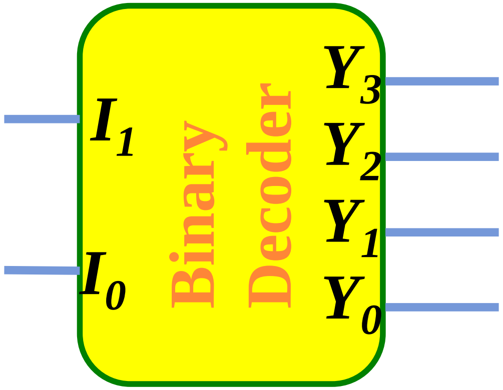


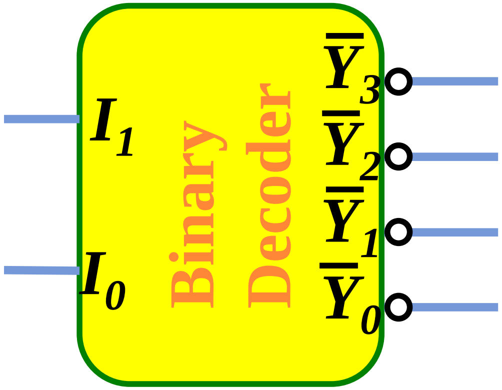
Decoders

- Active-High / Active-Low

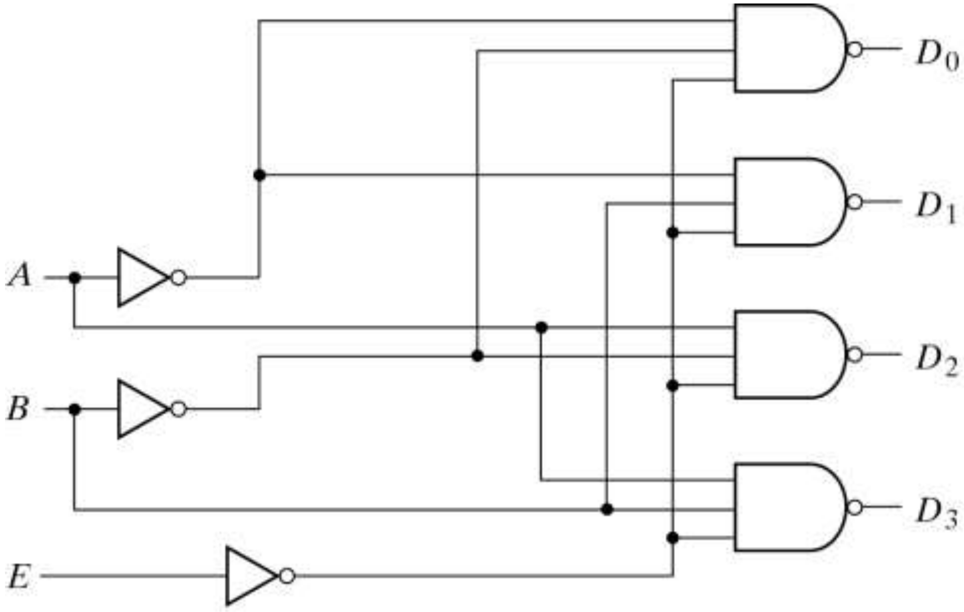
I_1 I_0	Y_3 Y_2 Y_1 Y_0
0 0	0 0 0 1
0 1	0 0 1 0
1 0	0 1 0 0
1 1	1 0 0 0

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





DECODERS



(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

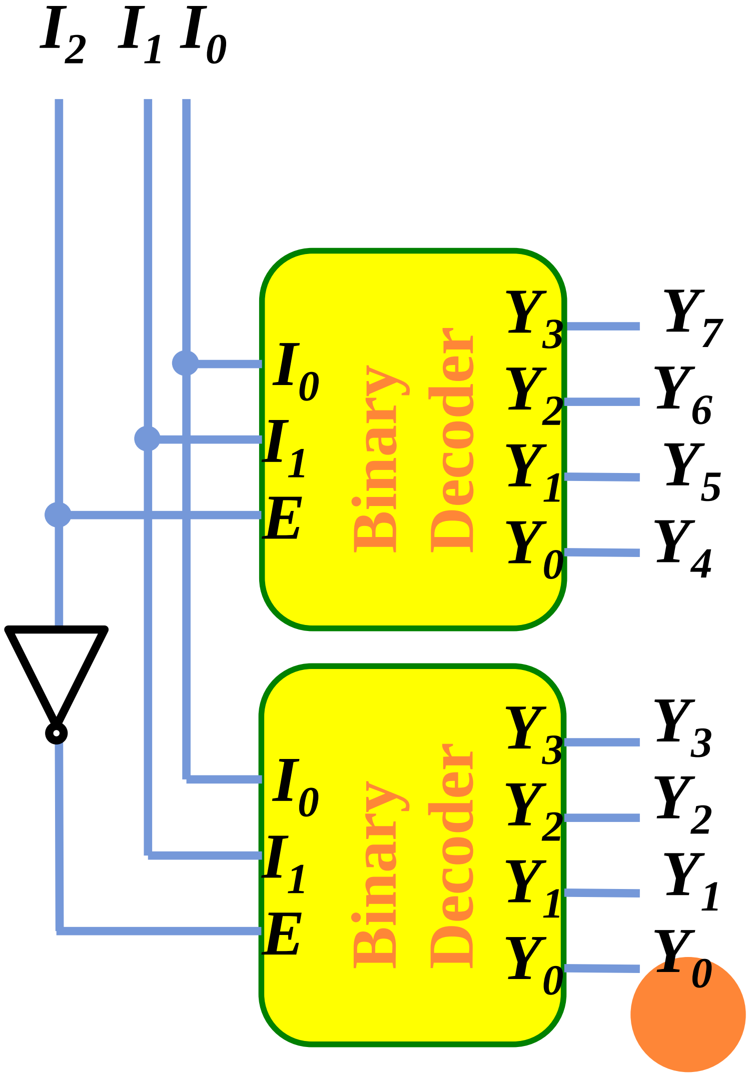
(b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

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



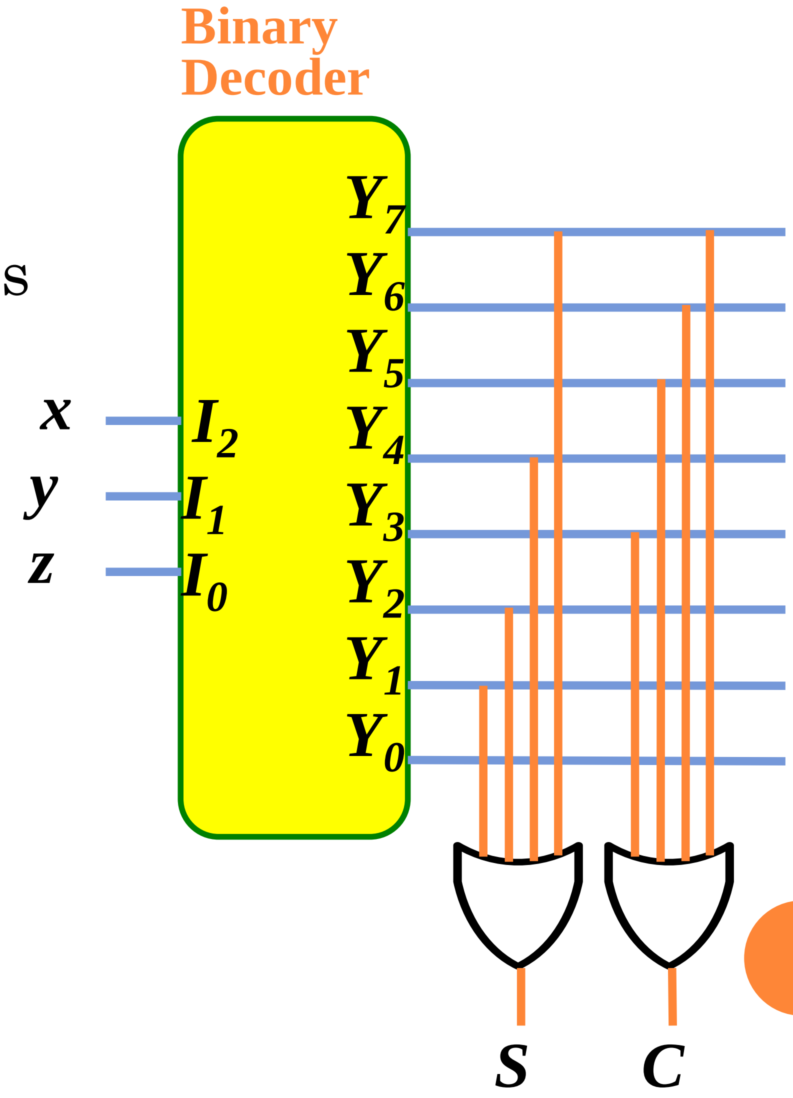
Implementation Using Decoders

- Each output is a minterm
- All minterms are produced
- 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

- Does reverse operation to decoder
- An encoder has 2^n (or fewer) input lines and n output lines
- Constraint – only one input is active at a time

ENCODERS

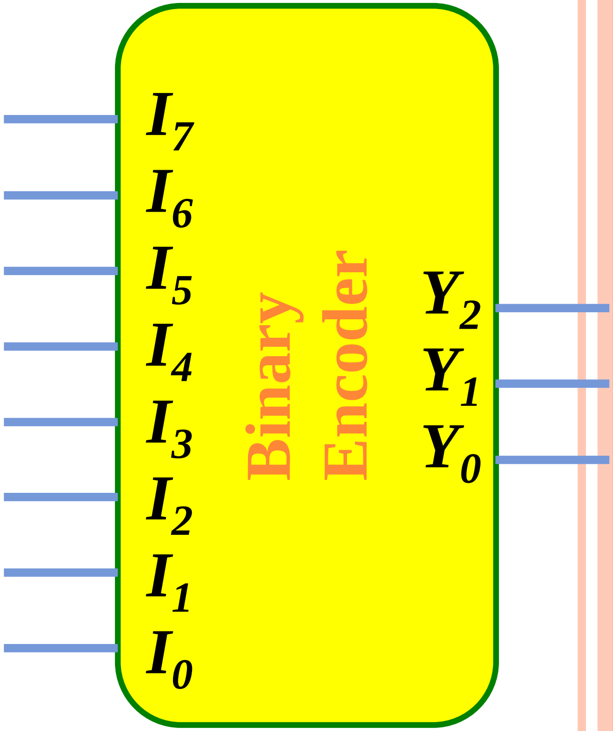
- Octal-to-Binary Encoder (8-to-3)

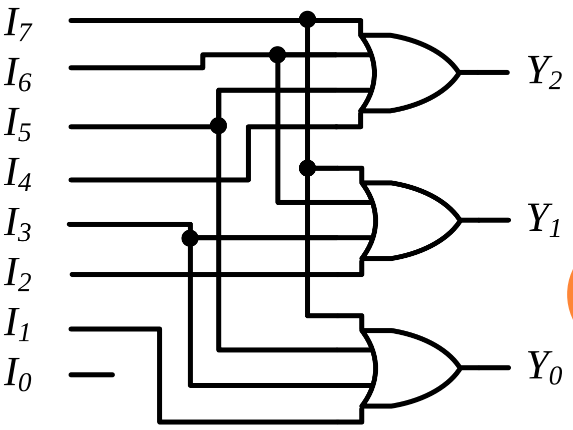
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

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

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

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





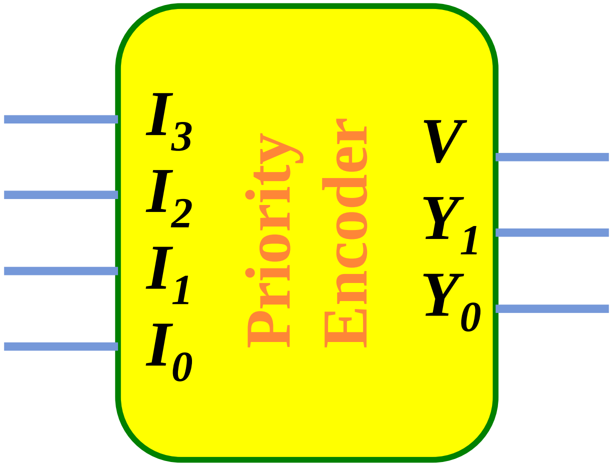
Priority Encoders

- Encoder with priority function
  - Multiple inputs may be true simultaneously
  - Higher priority input gets the precedence

Priority Encoders

- 4-Input Priority Encoder

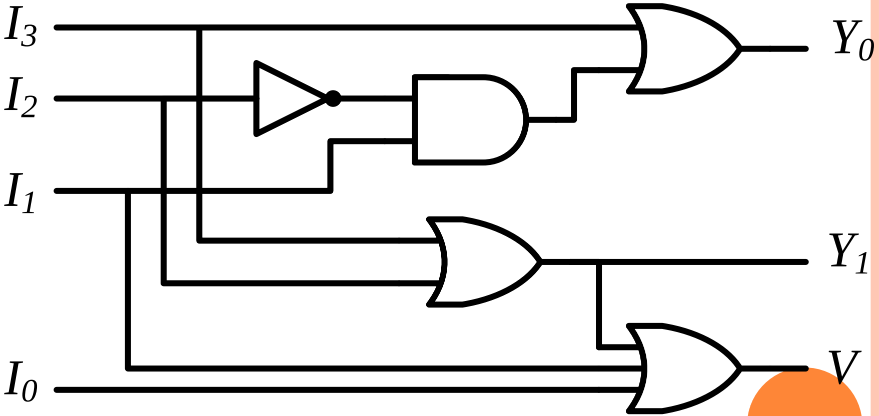
I_3	I_2	I_1	I_0	Y_1	Y_0	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1



$$Y_1 = I_3 + I_2$$

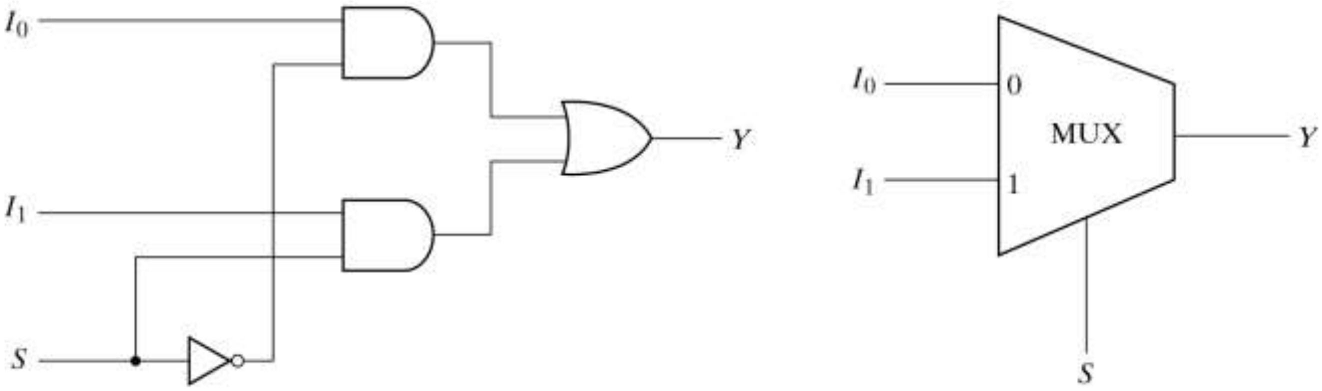
$$Y_0 = I_3 + \bar{I}_2 I_1$$

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



MULTIPLEXERS

- A multiplexer is a combinational circuit that selects one of many input lines (2^n) and directs it to its single output line.
- There are n selection lines whose bit combinations determine which input is selected.



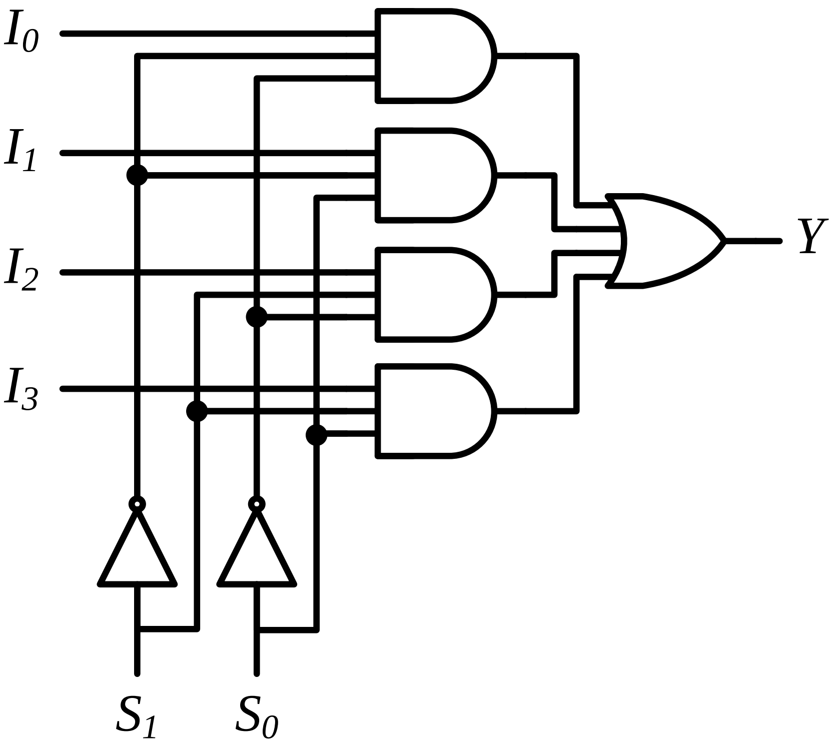
(a) Logic diagram

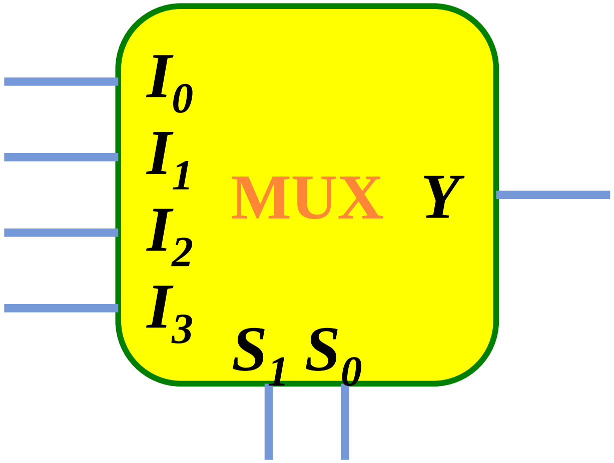
(b) Block diagram

Fig. 4-24 2-to-1-Line Multiplexer

Multiplexers

- 4-to-1 MUX

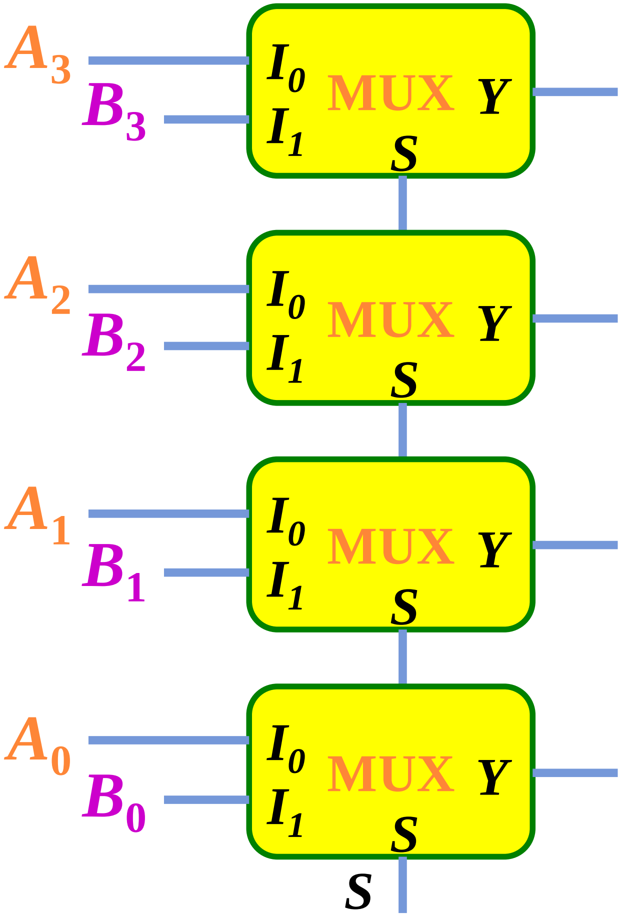


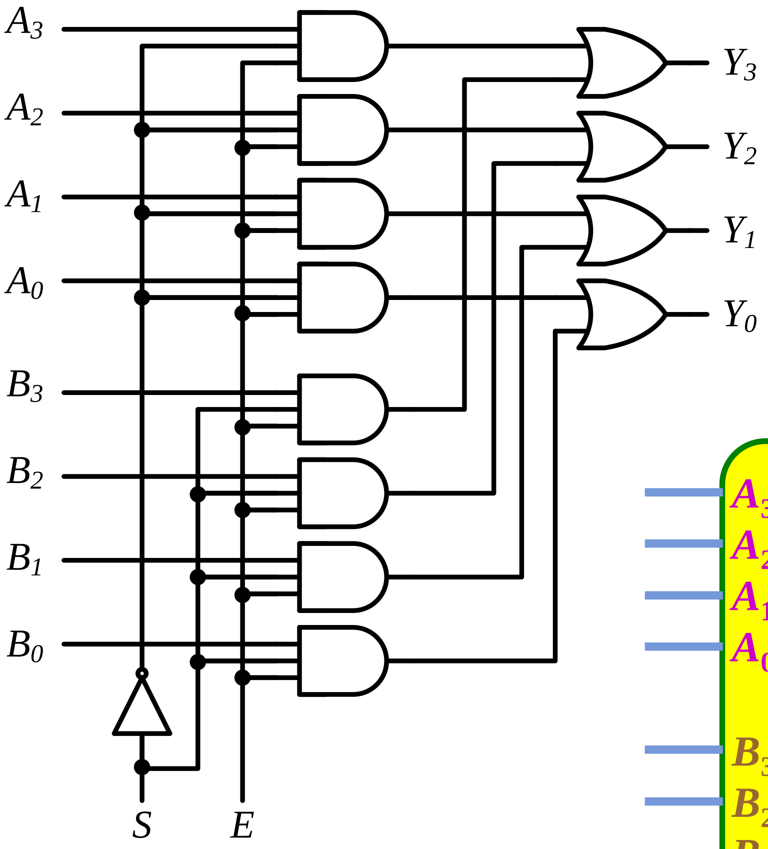


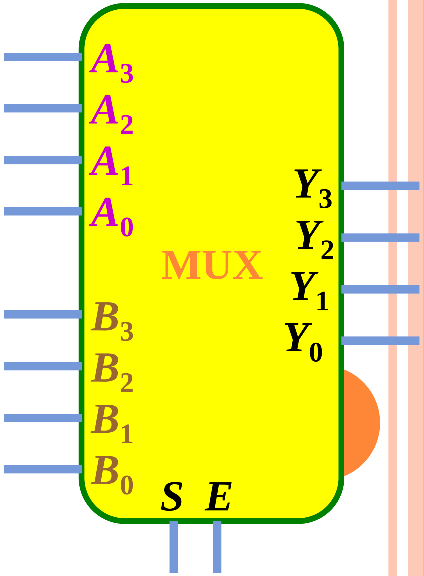
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Multiplexers

- Quad 2-to-1 MUX

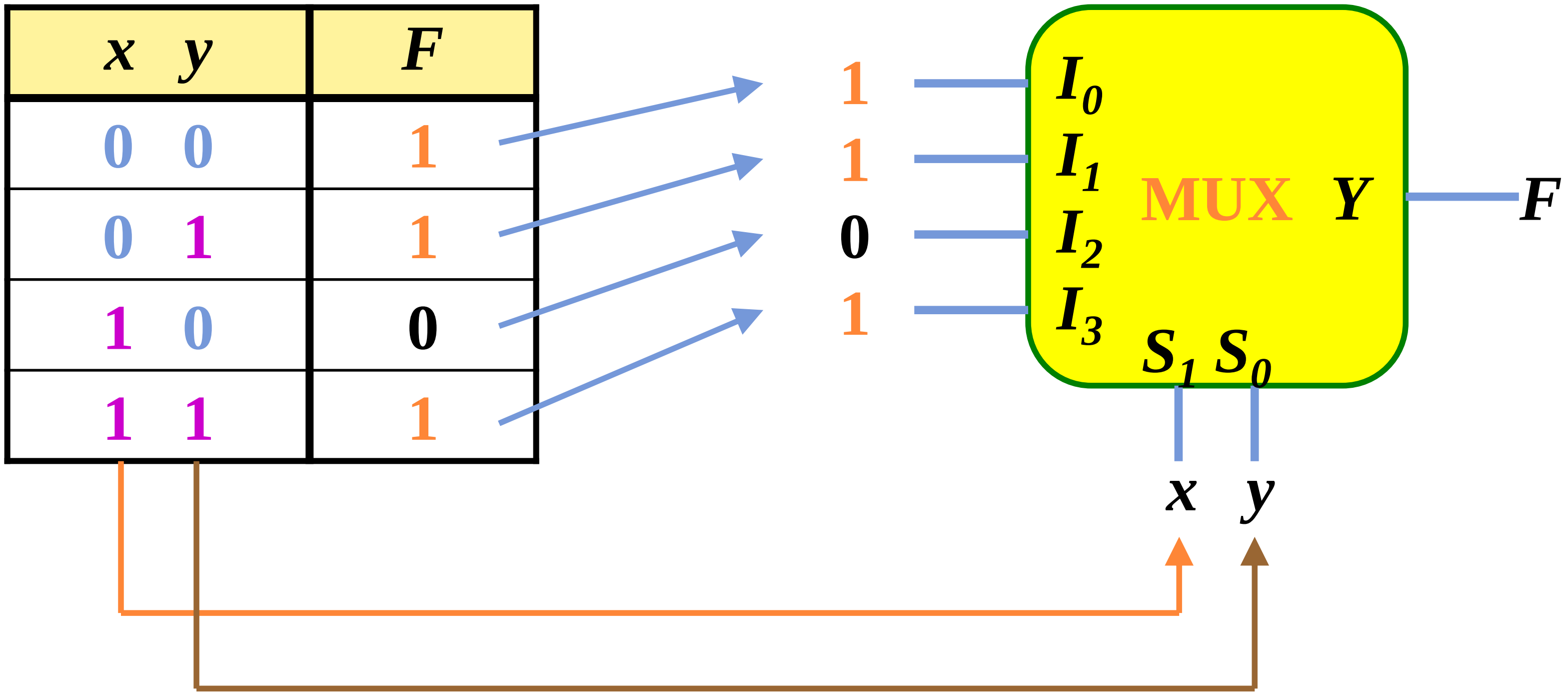






Implementation Using Multiplexers

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

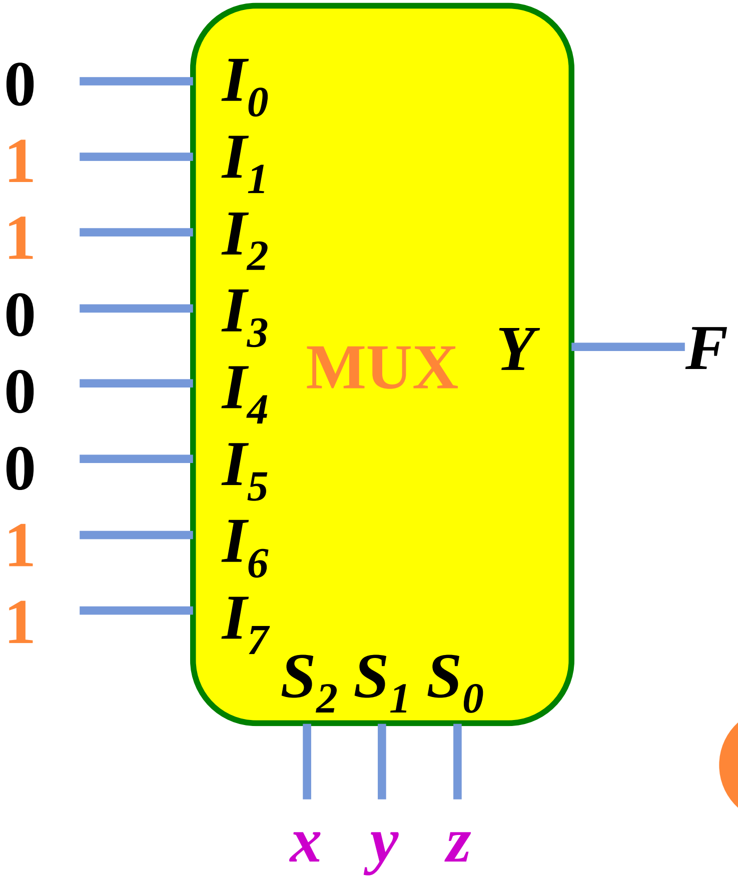


x y	F
0 0	1
0 1	1
1 0	0
1 1	1

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

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

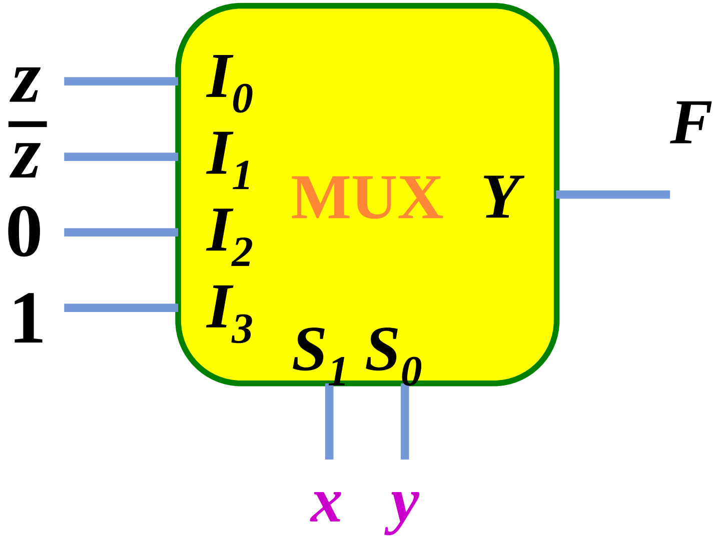
x	*y*	*z*	*F*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$F = z$

$F = \bar{z}$

$F = 0$

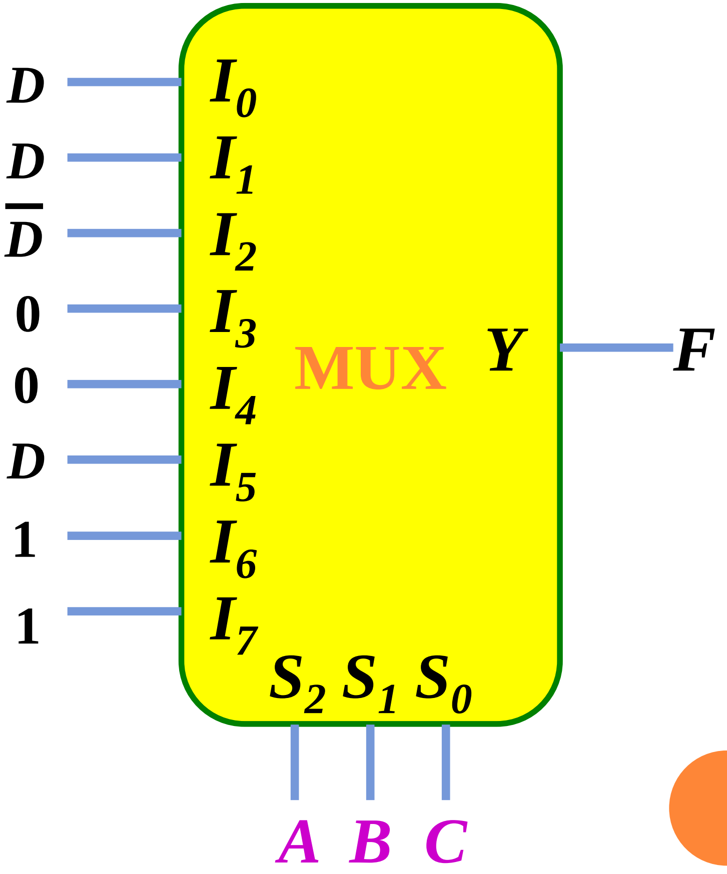
$F = 1$



Implementation Using Multiplexers

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

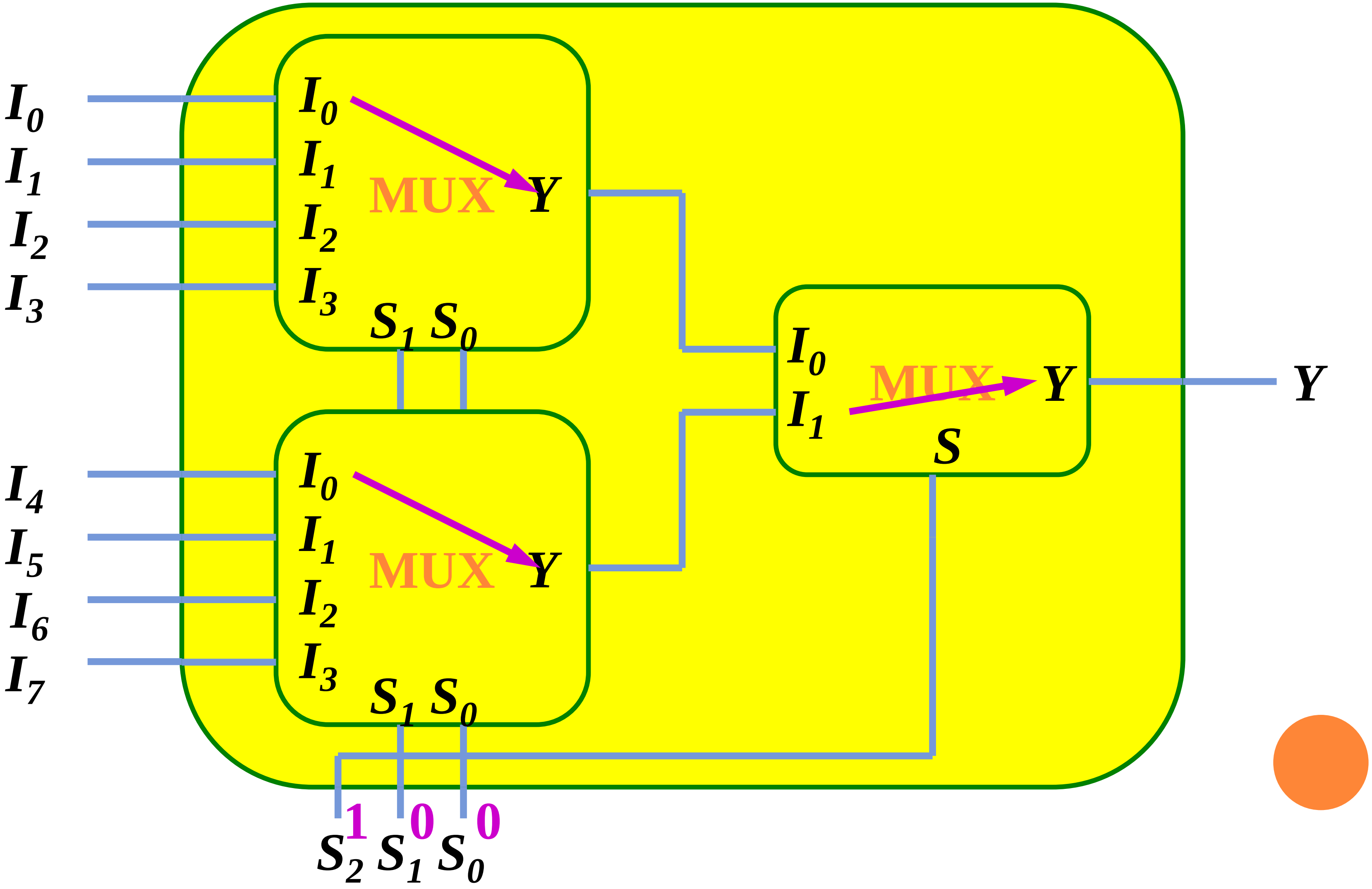
A B C D	F	
0 0 0 0	0	$F = D$
0 0 0 1	1	
0 0 1 0	0	$F = D$
0 0 1 1	1	
0 1 0 0	1	$F = \overline{D}$
0 1 0 1	0	
0 1 1 0	0	$F = 0$
0 1 1 1	0	
1 0 0 0	0	$F = 0$
1 0 0 1	0	
1 0 1 0	0	$F = D$
1 0 1 1	1	
1 1 0 0	1	$F = 1$
1 1 0 1	1	
1 1 1 0	1	$F = 1$
1 1 1 1	1	



Implementation Using Multiplexers

- Steps:
  - Complete the truth table from the SOP.
  - The first *n – 1 variables in the table are applied to the* selection inputs of the multiplexer.
  - For each combination of the selection variables, we evaluate the output as a function of the last variable.
  - Apply these values to the data input in proper order.

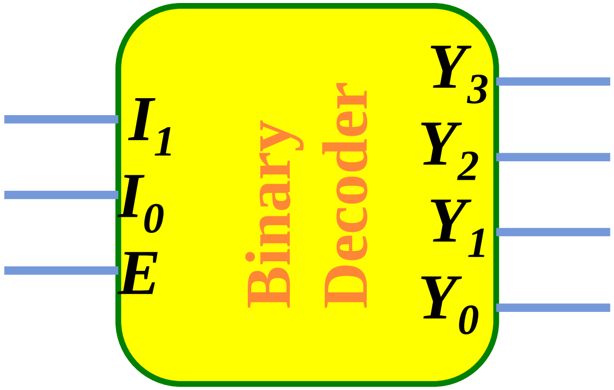
Multiplexer Expansion
8-to-1 MUX using Dual 4-to-1 MUX

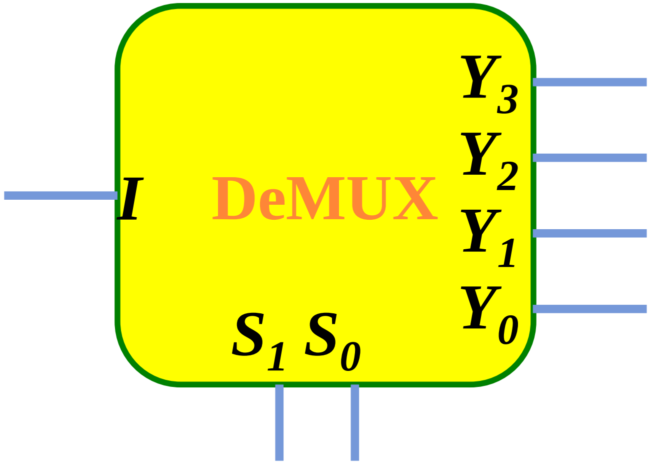


DeMultiplexers

- A decoder with enable input can function as a demultiplexer – a circuit receives information from a single line and directs it to one of 2^n possible output lines.

DeMultiplexers / Decoders

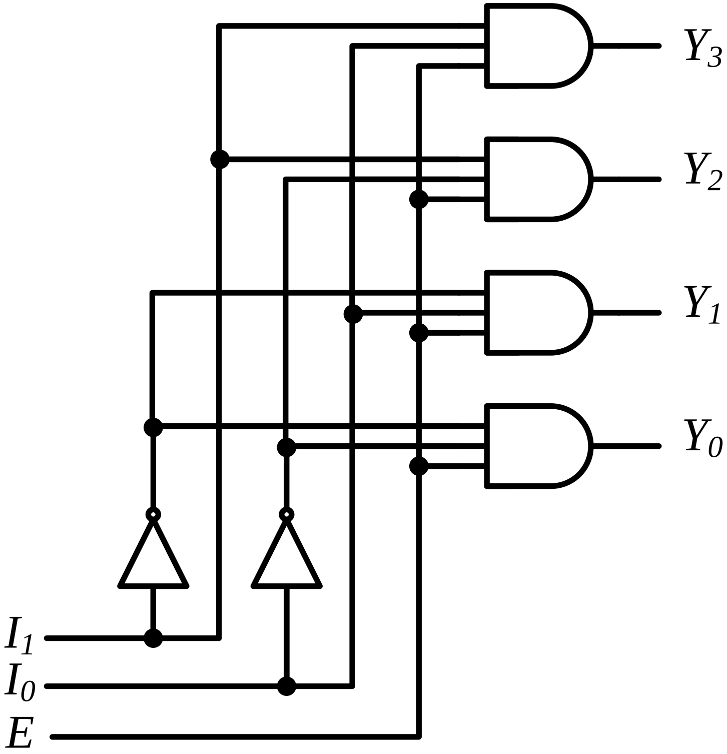




E	I_1 I_0	Y_3 Y_2 Y_1 Y_0
0	x x	0 0 0 0
1	0 0	0 0 0 1
1	0 1	0 0 1 0
1	1 0	0 1 0 0
1	1 1	1 0 0 0

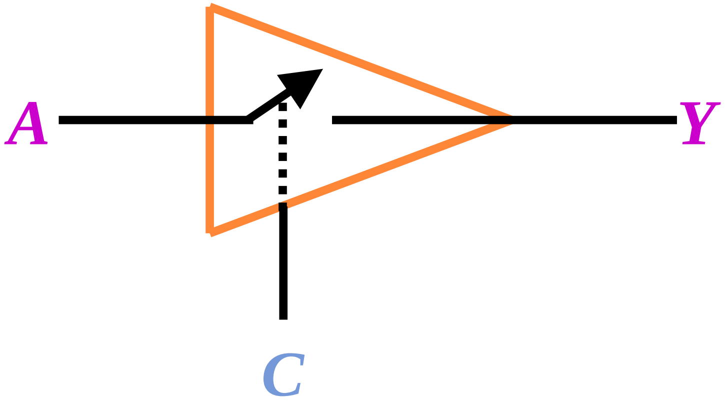
Demultiplexers

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

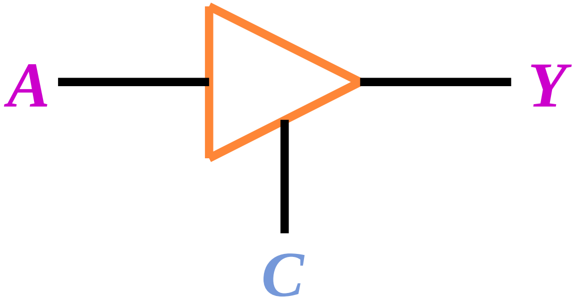


Three-State Gates

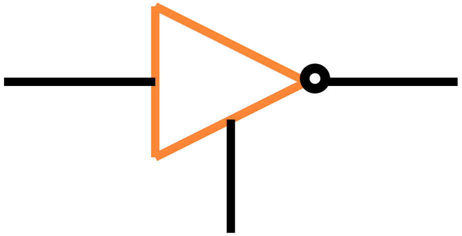
- Tri-State Buffer



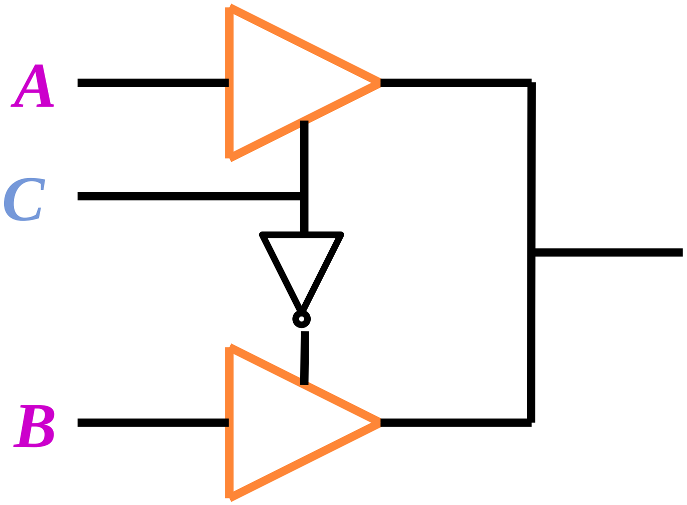
C A	Y
0 x	Hi-Z
1 0	0
1 1	1



- Tri-State Inverter

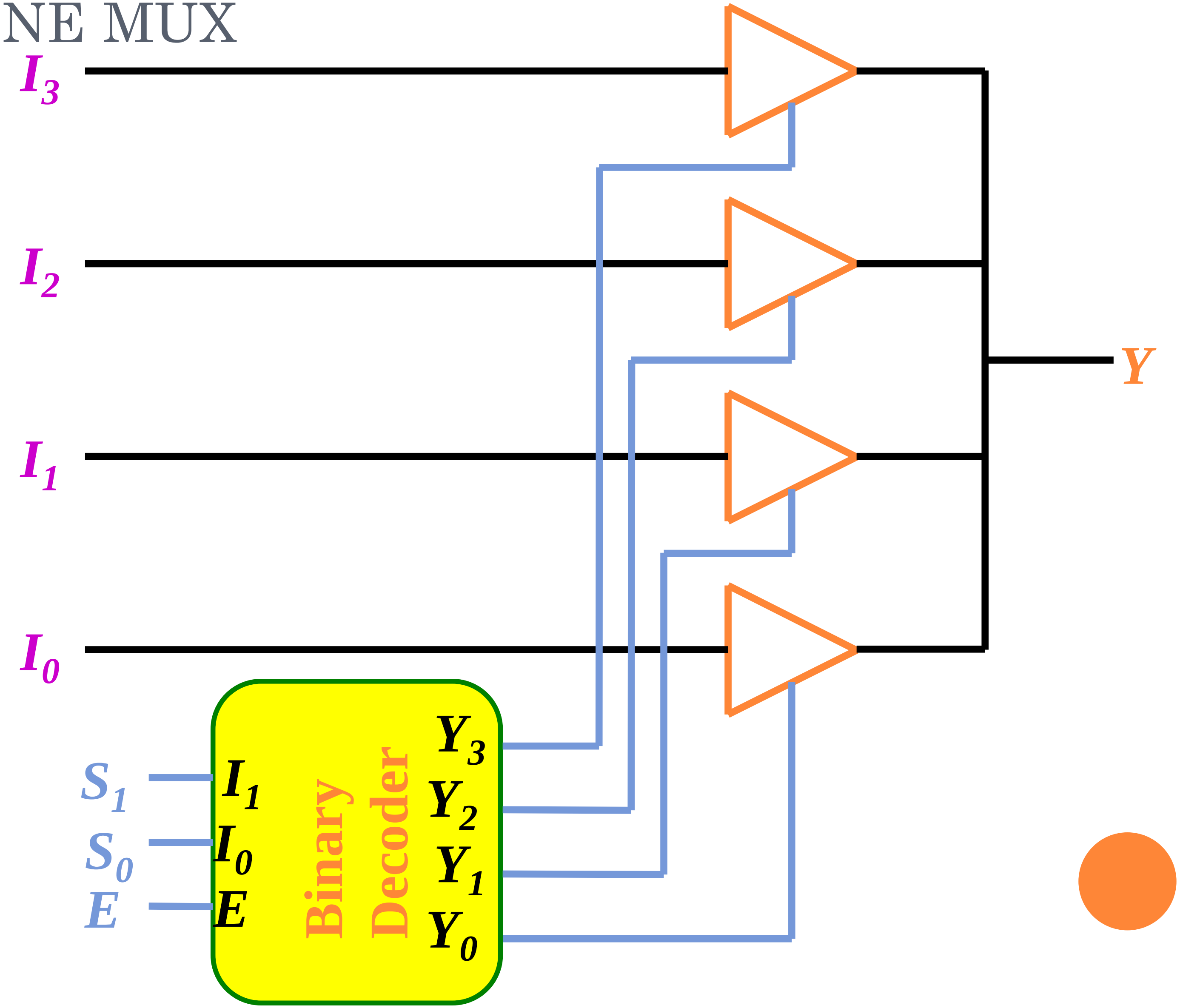


Three-State Gates
2-to-1-line MUX



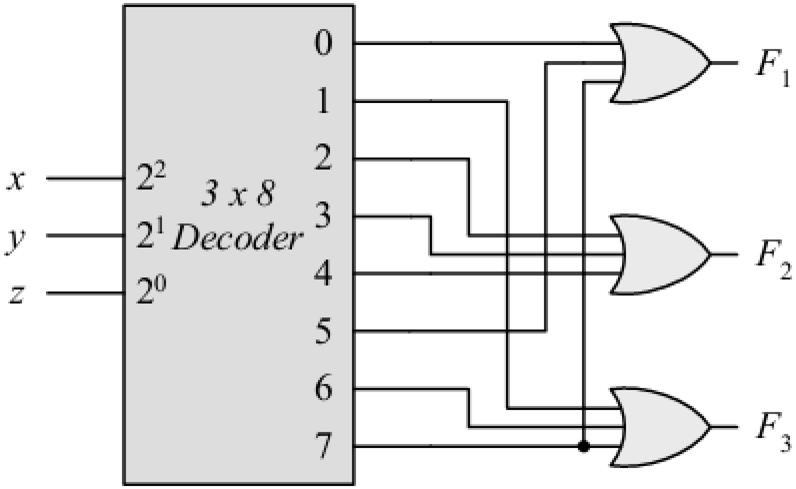
$$Y = \begin{cases} A & \text{if } C = 1 \\ B & \text{if } C = 0 \end{cases}$$

Three-State Gates
4-to-1-line MUX



Practice

- Using a decoder and external gates, design the combinational circuit defined by the following three boolean functions:
- $F_1 = x'y'z' + xz = \sum(0, 5, 7)$
- $F_2 = xy'z' + x'y = \sum(2, 3, 4)$
- $F_3 = x'y'z + xy = \sum(1, 6, 7)$



PRACTICE

- A combinational circuit is specified by the following three boolean functions:

  $F_1(A, B, C) = \sum (2, 4, 7)$

  $F_2(A, B, C) = \sum (0, 3)$

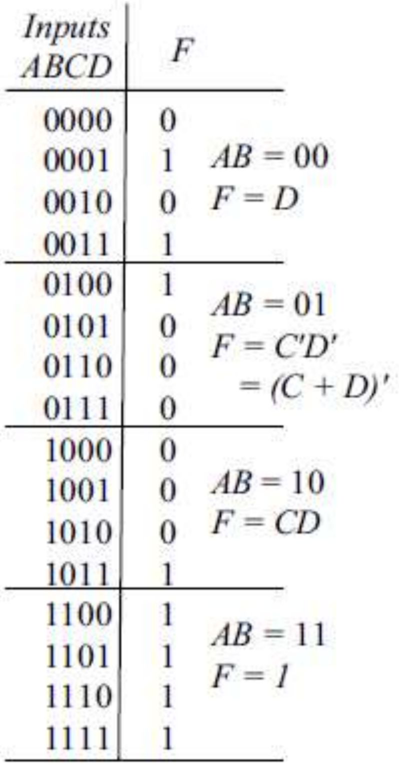
  $F_3(A, B, C) = \sum (0, 2, 3, 4, 7)$

- Implement the circuit with a decoder constructed with NAND gates.

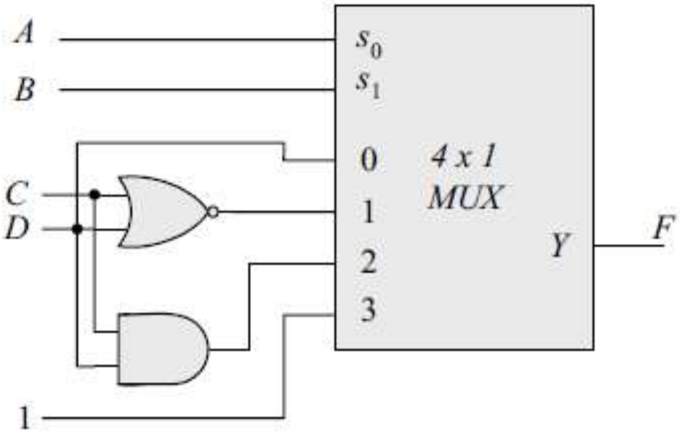
PRACTICE

- Implement the following Boolean function with a 4 X 1 multiplexer and external gates.

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



Inputs ABCD	F	
0000	0	
0001	1	$AB = 00$
0010	0	$F = D$
0011	1	
0100	1	
0101	0	$AB = 01$
0110	0	$F = C'D'$
0111	0	$= (C + D)'$
1000	0	
1001	0	$AB = 10$
1010	0	$F = CD$
1011	1	
1100	1	
1101	1	$AB = 11$
1110	1	$F = 1$
1111	1	

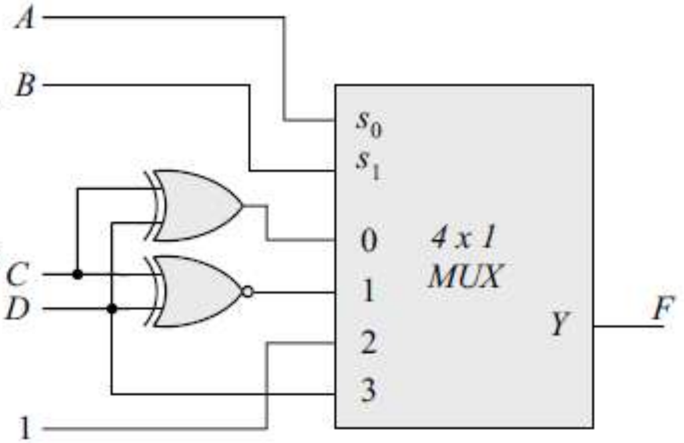


Practice

- Implement the following Boole an function with a 4 X 1 multiplexer and external gates.

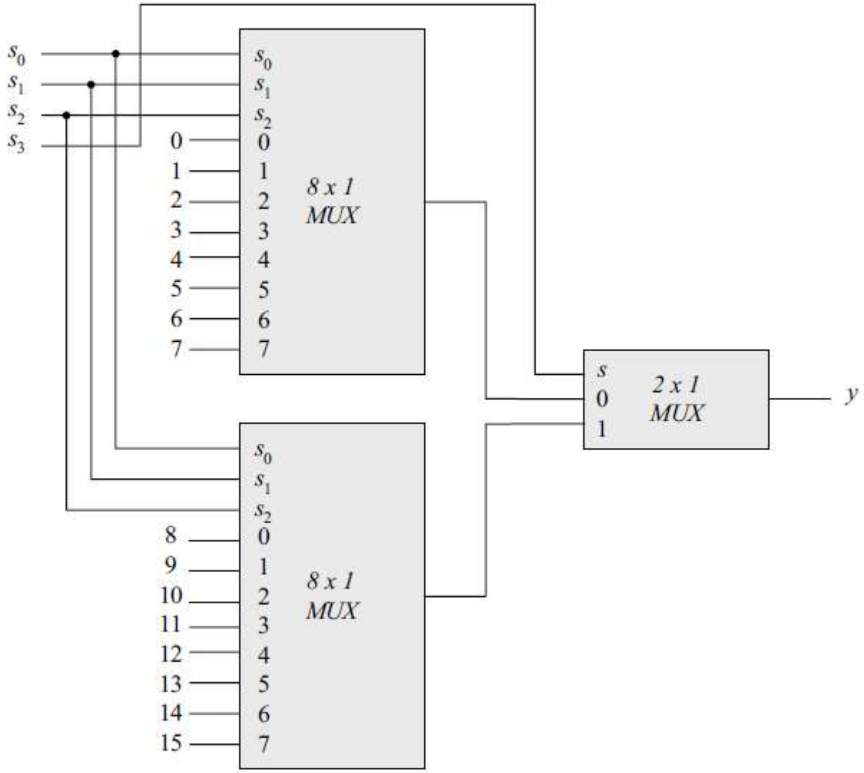
  $F(A, B, C, D) = \sum (1, 2, 4, 7, 8, 9, 10, 11, 13, 15)$

Inputs ABCD	F	
0000	0	
0001	1	$AB = 00$
0010	1	$F = C'D + CD'$
0011	0	
0100	1	
0101	0	$AB = 01$
0110	0	$F = C'D' + CD$
0111	1	
1000	1	
1001	1	$AB = 10$
1010	1	$F = 1$
1011	1	
1100	0	$AB = 11$
1101	1	$F = D$
1110	0	
1111	1	



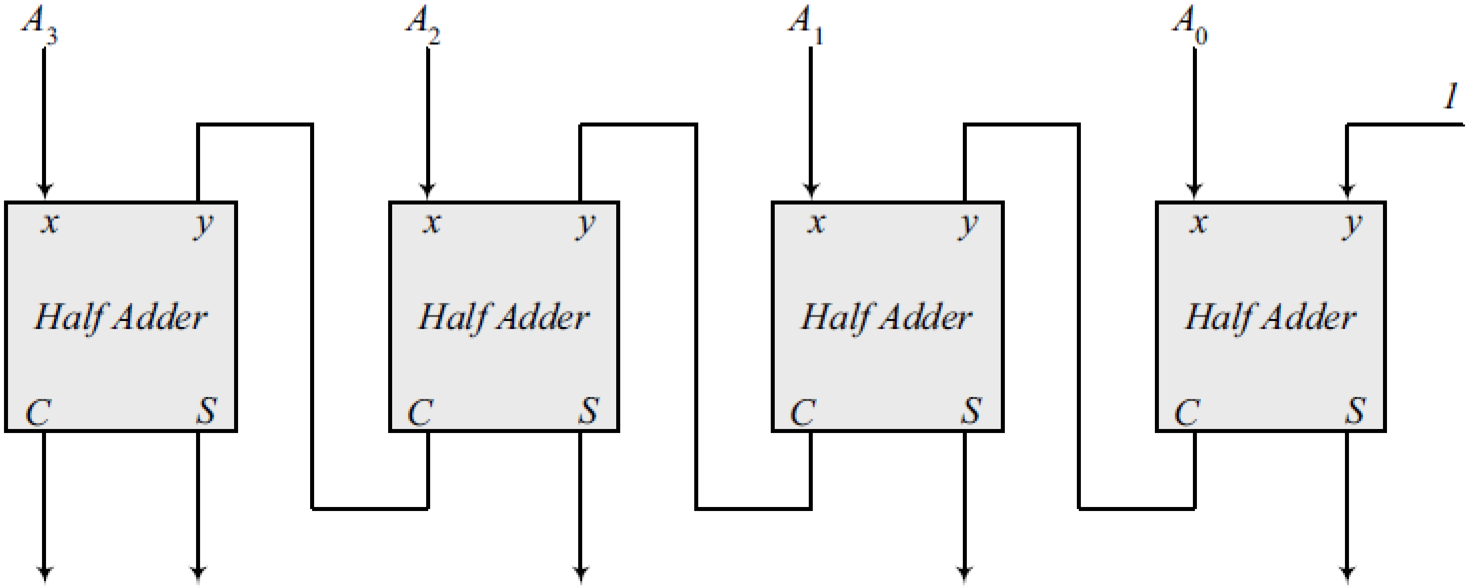
PRACTICE

- Construct a 16 X 1 multiplexer with two 8 X 1 and one 2 X 1 multiplexers. Use block diagrams



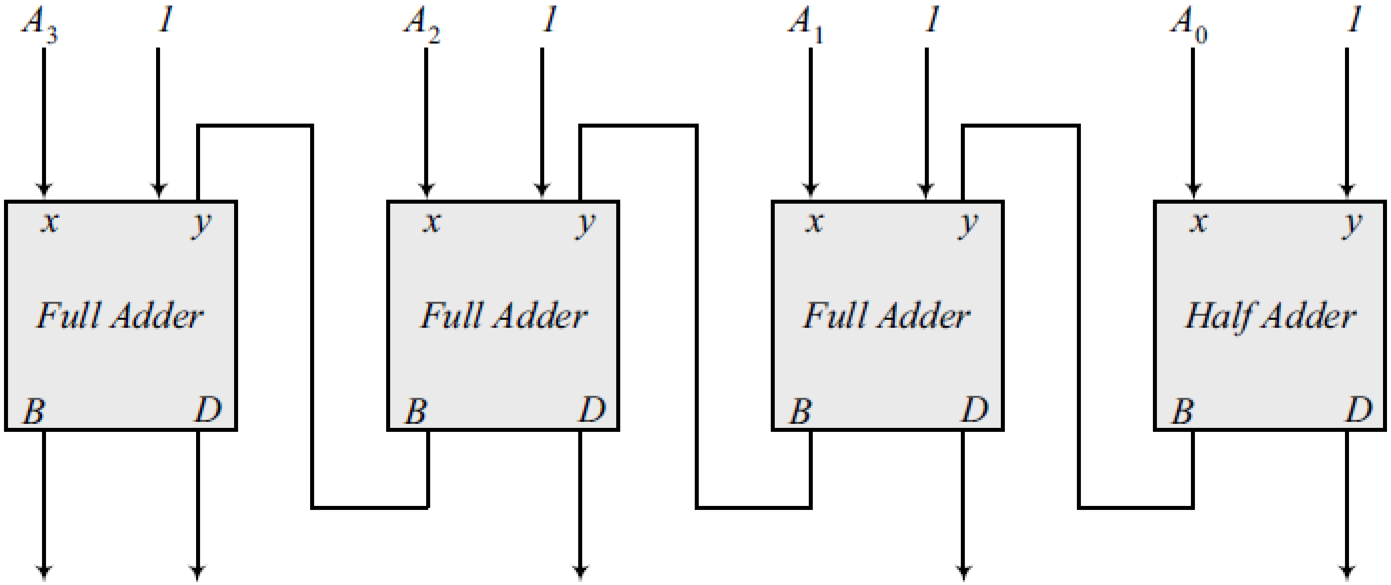
Practice

- Using four half-adders design a 4-bit combinational circuit incrementer (a circuit that adds 1 to a 4-bit binary number)



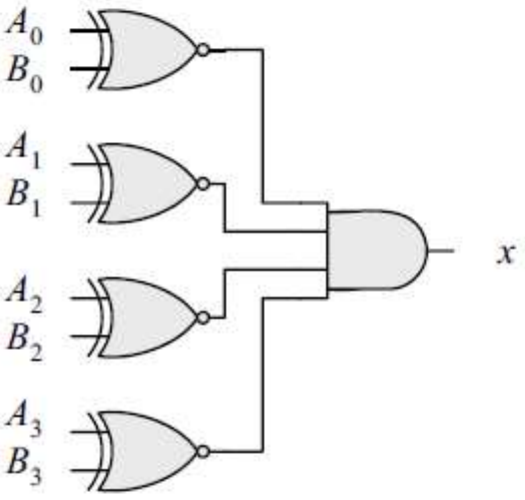
PRACTICE

- Using a half-adder and three full-adeders design a 4-bit combinational circuit decrementer (a circuit that subtracts 1 from a 4-bit binary number)



Practice

- Design a combinational circuit that compares two 4-bit numbers to check if they are equal. The circuit output is equal to 1 if two numbers are equal and 0 otherwise.



$$x = (A_0 \oplus B_0)'(A_1 \oplus B_1)'(A_2 \oplus B_2)'(A_3 \oplus B_3)'$$