Digital Electronics and Microprocessors

Class 16

CHHAYADEVI BHAMARE

MSI logic circuits(Chapter 9 of T1)

- Digital systems obtain data and information continuously operated on in some manner:
  - *Decoding/encoding.*
  - *Multiplexing/demultiplexing,.*
  - *Comparison; Code conversion;*
- These and other operations have been facilitated by the availability of numerous ICs in the MSI (medium-scale-integration) category.

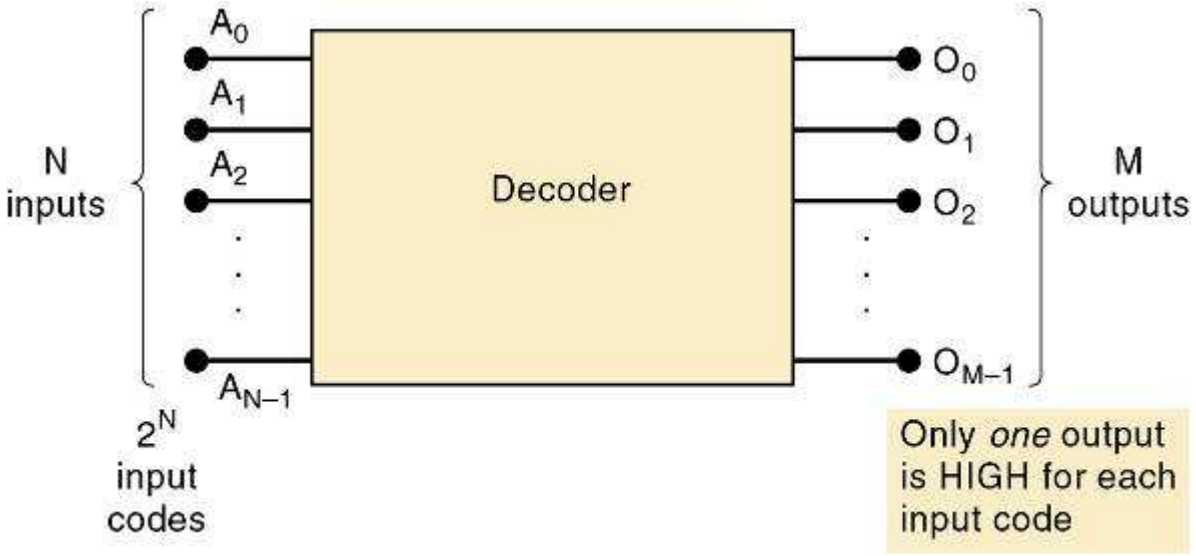
Decoders

- Decoders are used when an output or a group of outputs is to be activated only on the occurrence of a specific combination of input levels.
  - Often provided by outputs of a counter or a register.

Decoders

- A **decoder** accepts a set of inputs that represents a binary number—activating only the output that corresponds to the input number.

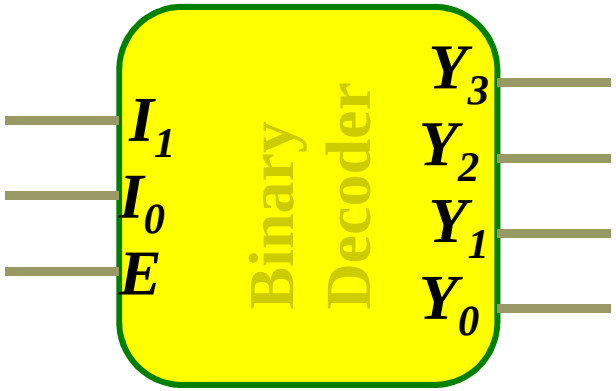
For each of these input combinations, only one of the *M* outputs will be active (HIGH); all the other outputs are LOW.



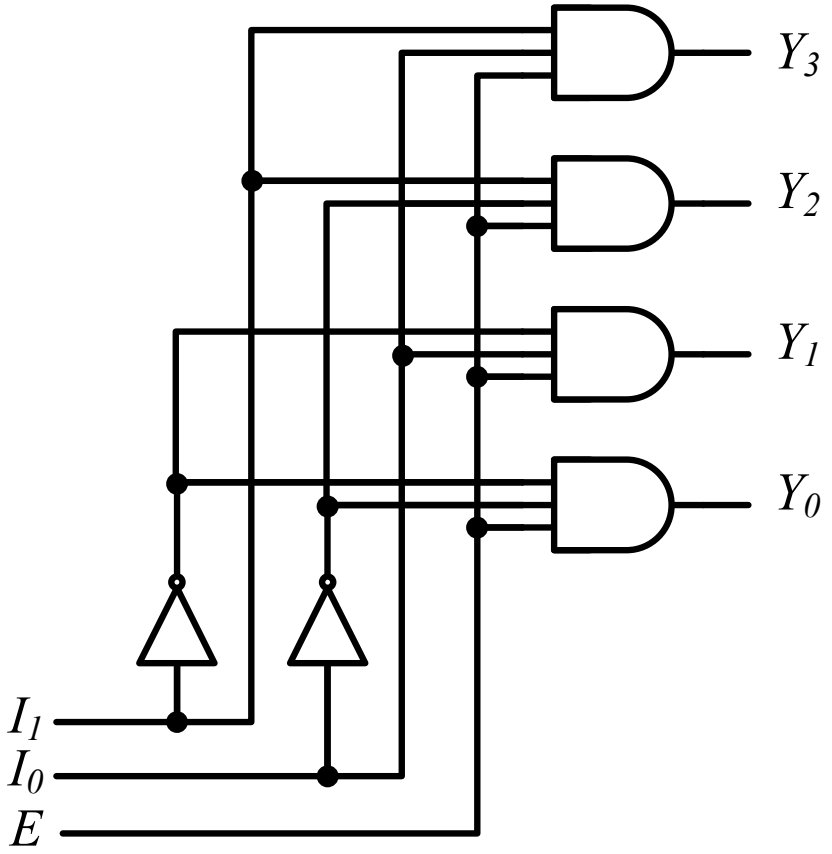
Many decoders are designed to produce active-LOW outputs, where only the selected output is LOW while all others are HIGH.

Decoders

- "*Enable*" Control



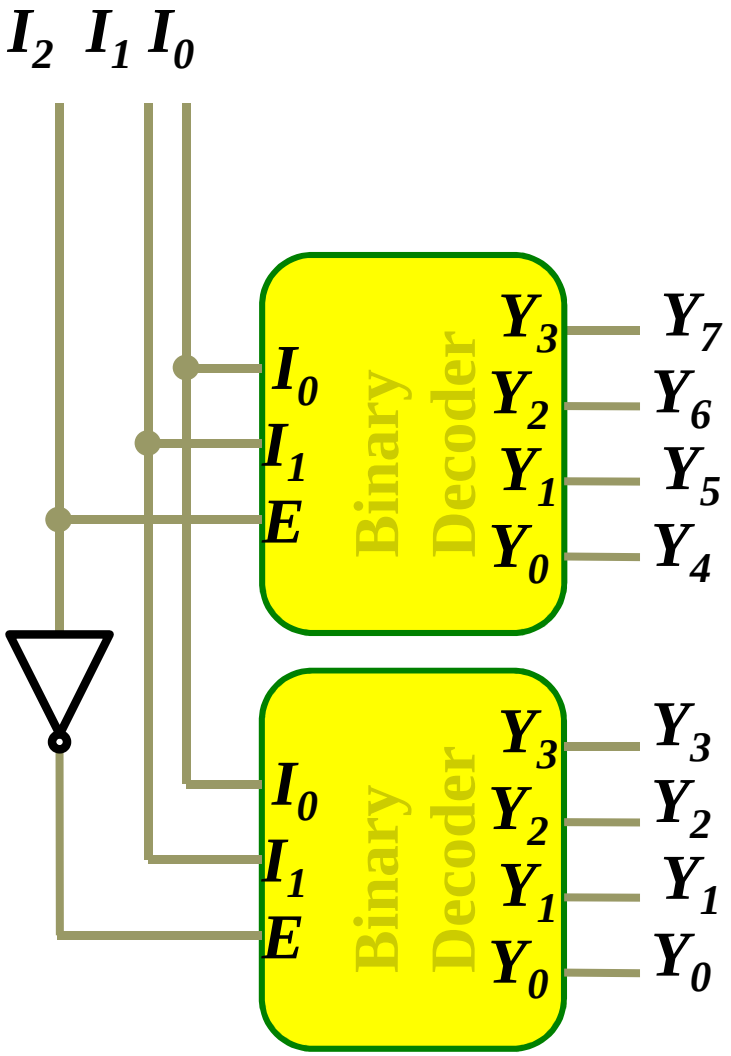
E	I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



Decoders

- 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

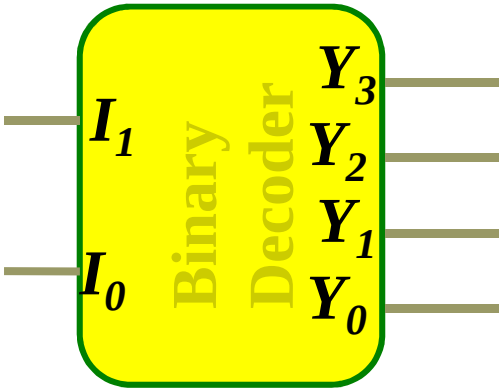


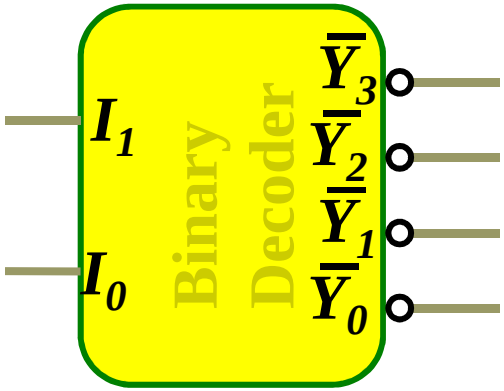
Decoders

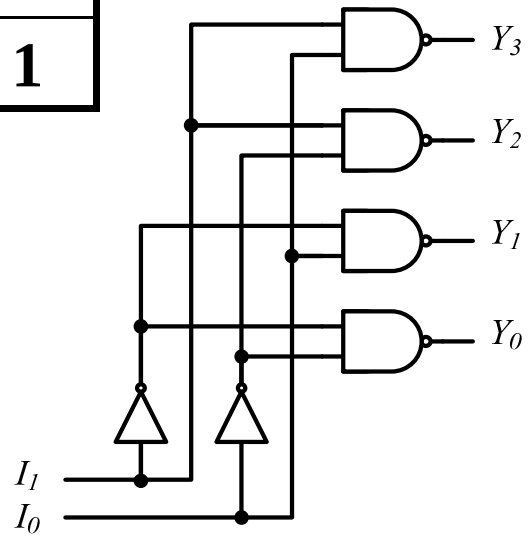
- Active-High / Active-Low

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







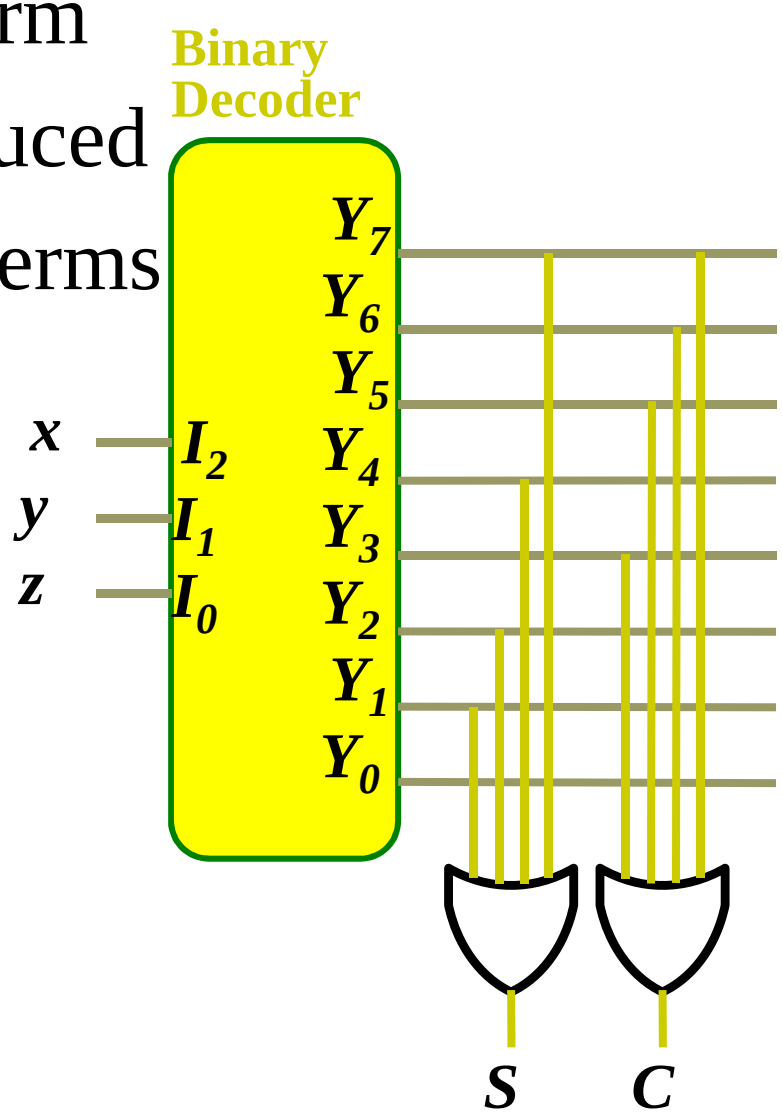
Implementation Using Decoders

- Each output is a minterm
- All minterms are produced
- Sum the required minterms

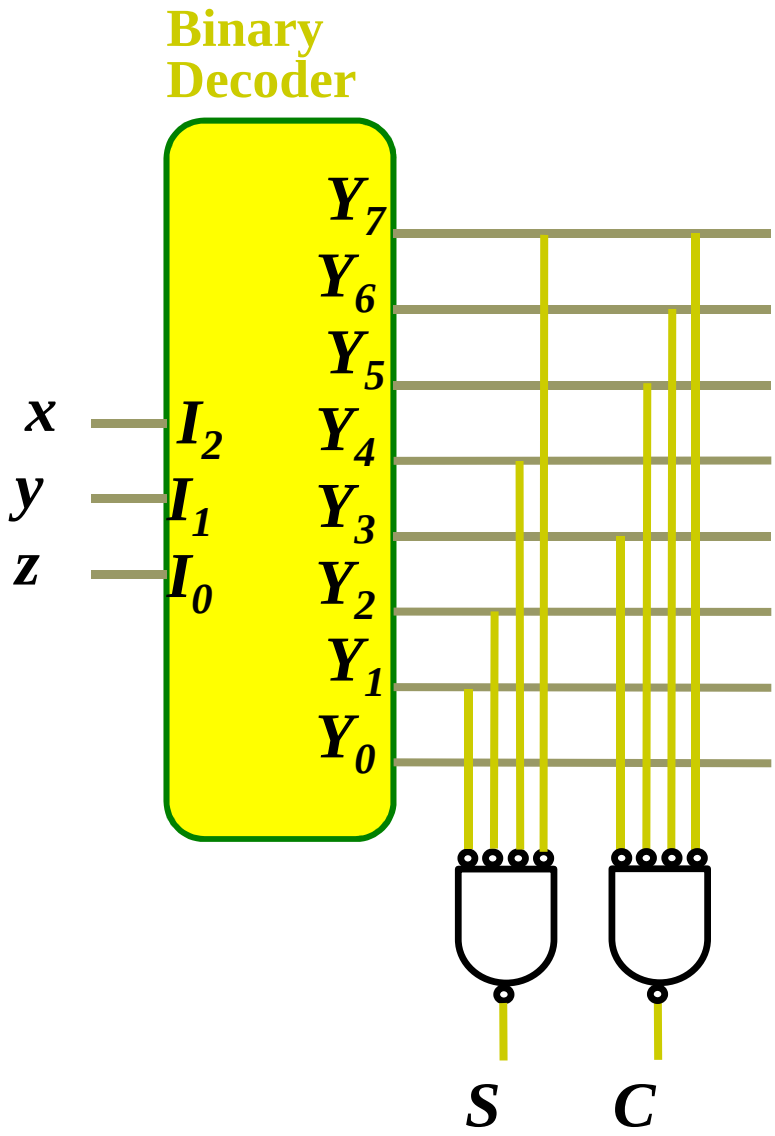
Example: Full Adder

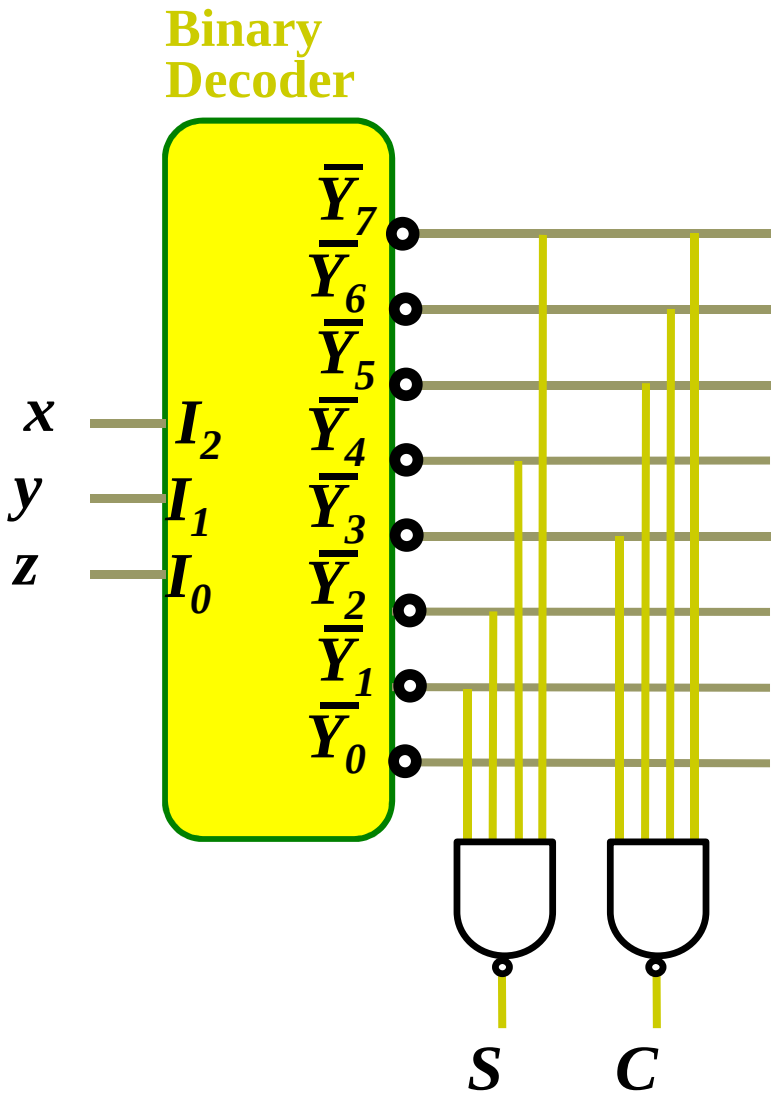
$S(x, y, z) = \sum(1, 2, 4, 7)$

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



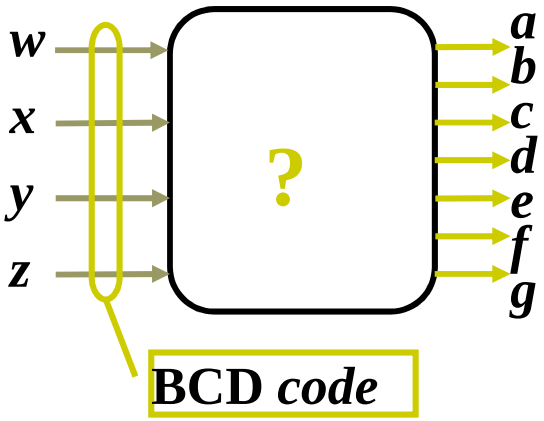
Implementation Using Decoders

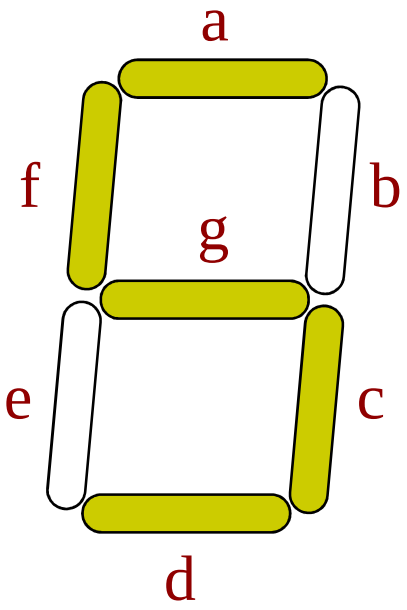


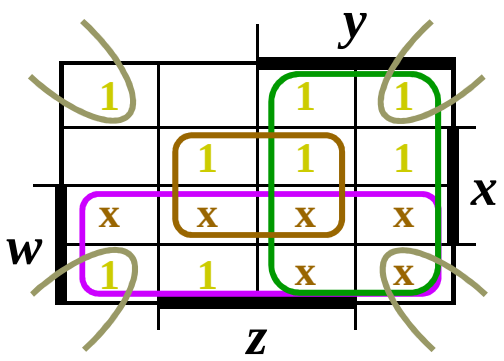


Seven-Segment Decoder

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







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

$b = \ldots$
$c = \ldots$
$d = \ldots$

Encoders

- Most decoders accept an input code & produce a HIGH (or LOW) at *one* and *only one* output line.
  - A decoder identifies, recognizes, or detects a particular code.

Encoders

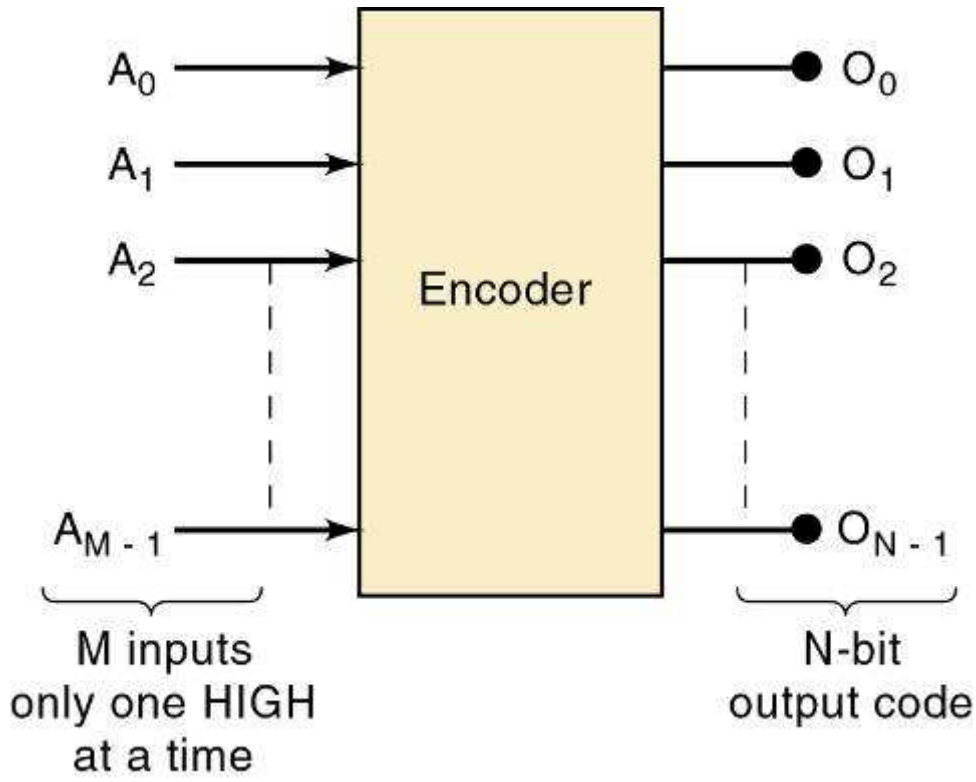
- The opposite of decoding process is **encoding**.
  - Performed by a logic circuit called an **encoder**.

An encoder has a number of input lines, only **one** of which is activated at a given time.

Shown is an encoder with *M* inputs and *N* outputs.

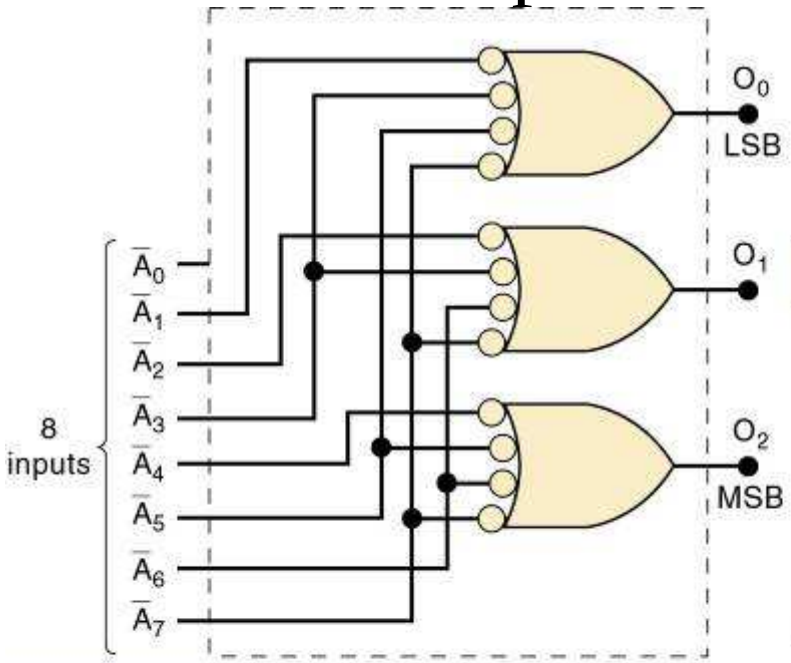
Inputs are active-HIGH, which means that they are normally LOW.

It produces an *N*-bit output code, depending on which input is activated.



Encoders

- An *octal-to-binary encoder (8-line-to-3-line encoder)* accepts eight input lines, producing a three-bit output code corresponding to the



Inputs								Outputs		
$\overline{A}_0$	$\overline{A}_1$	$\overline{A}_2$	$\overline{A}_3$	$\overline{A}_4$	$\overline{A}_5$	$\overline{A}_6$	$\overline{A}_7$	O_2	O_1	O_0
X	1	1	1	1	1	1	1	0	0	0
X	0	1	1	1	1	1	1	0	0	1
X	1	0	1	1	1	1	1	0	1	0
X	1	1	0	1	1	1	1	0	1	1
X	1	1	1	0	1	1	1	1	0	0
X	1	1	1	1	0	1	1	1	0	1
X	1	1	1	1	1	0	1	1	1	0
X	1	1	1	1	1	1	0	1	1	1

*Only one LOW input at a time

Logic circuit for an octal-to-binary (8-line-to-3-line) encoder. Only one input should be active at one time.

Encoders

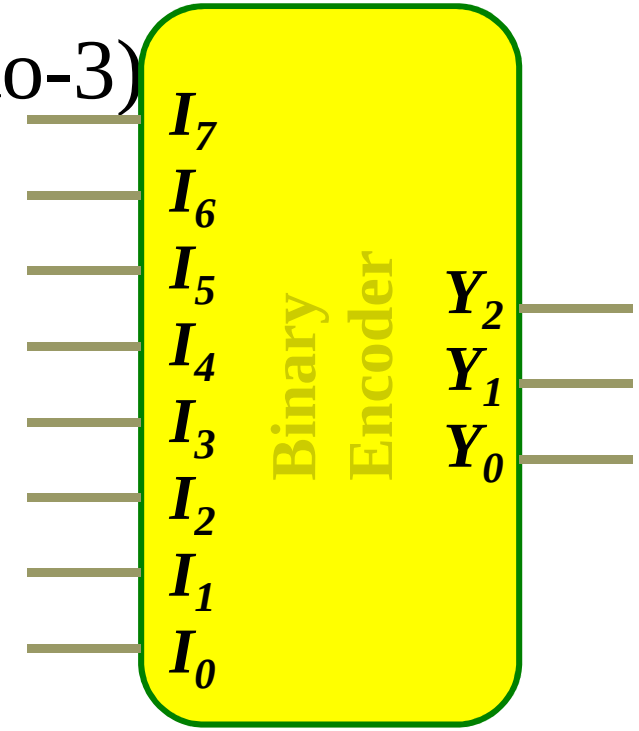
- Octal-to-Binary Encoder (8-to-3)

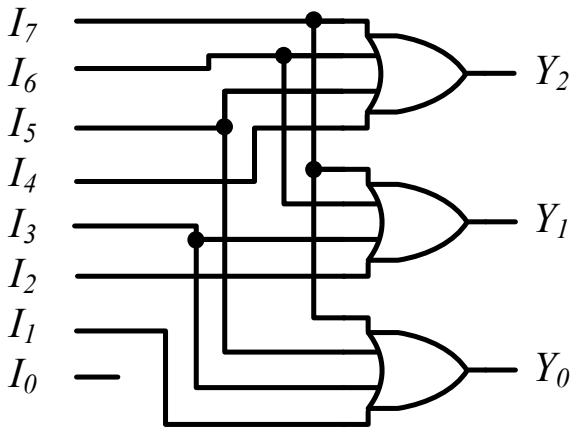
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

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

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

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





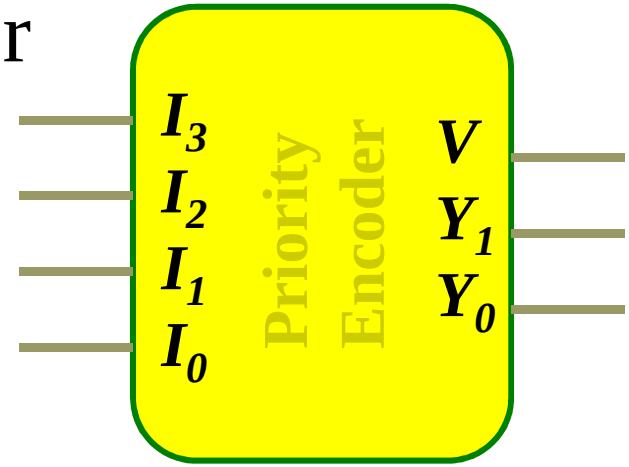
Priority Encoder

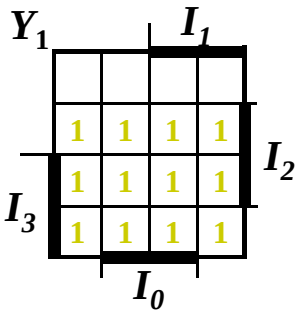
- A priority encoder is an encoder circuit that includes priority function.
- It means if two or more inputs are equal to 1 at the same time, the input having higher subscript number, considered as a higher priority. For example if D3 is 1 regardless of the value of the other input lines the result of output is 3 which is 11.
- If all inputs are 0, there is no valid input. For detecting this situation we considered a third output named V. V is equal to 0 when all input are 0 and is one for rest of the situations of TT.

Priority Encoders

- 4-Input Priority Encoder

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

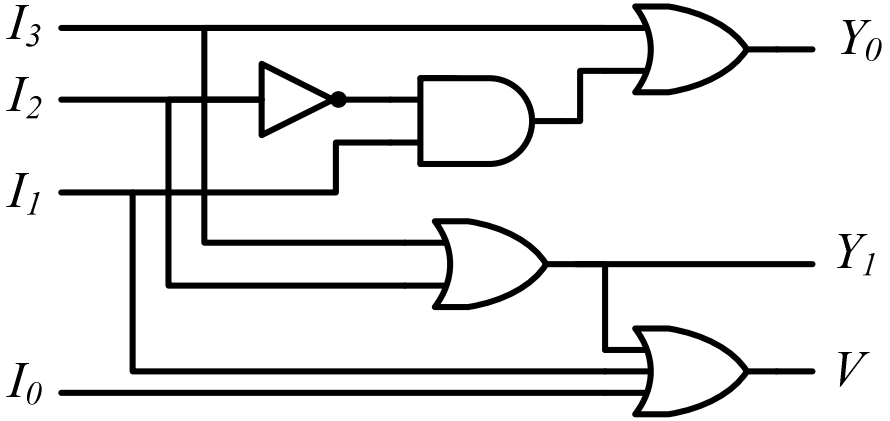




$$Y_1 = I_3 + I_2$$

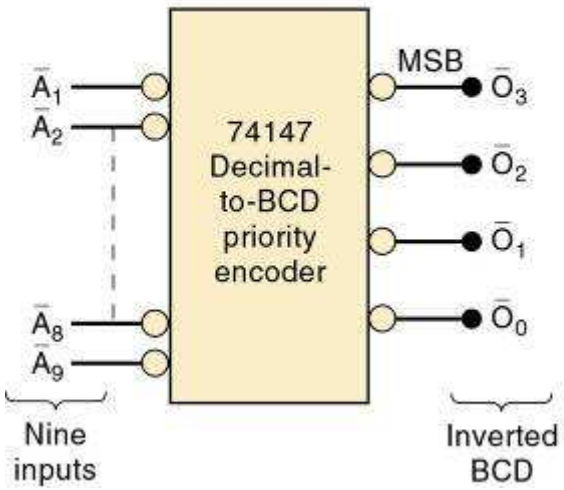
$$Y_0 = I_3 + \bar{I}_2 I_1$$

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



Encoders

- A **priority encode**r ensures that when two or more inputs are activated, the output code will correspond to the highest-numbered input.



$\bar{A}_1$	$\bar{A}_2$	$\bar{A}_3$	$\bar{A}_4$	$\bar{A}_5$	$\bar{A}_6$	$\bar{A}_7$	$\bar{A}_8$	$\bar{A}_9$	$\bar{O}_3$	$\bar{O}_2$	$\bar{O}_1$	$\bar{O}_0$
1	1	1	1	1	1	1	1	1	1	1	1	1
X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

X = either 0 or 1

It has nine active-LOW inputs represent decimal digits 1 through 9, producing *inverted* BCD code corresponding to the highest-numbered activated input.

Can u implement logic function using Encoders?

- Work for you

Encoders

- A **switch encoder** can be used when BCD data must be entered manually into a digital system.
  - The 10 switches might be the keyboard switches on a calculator—representing digits 0 through 9.

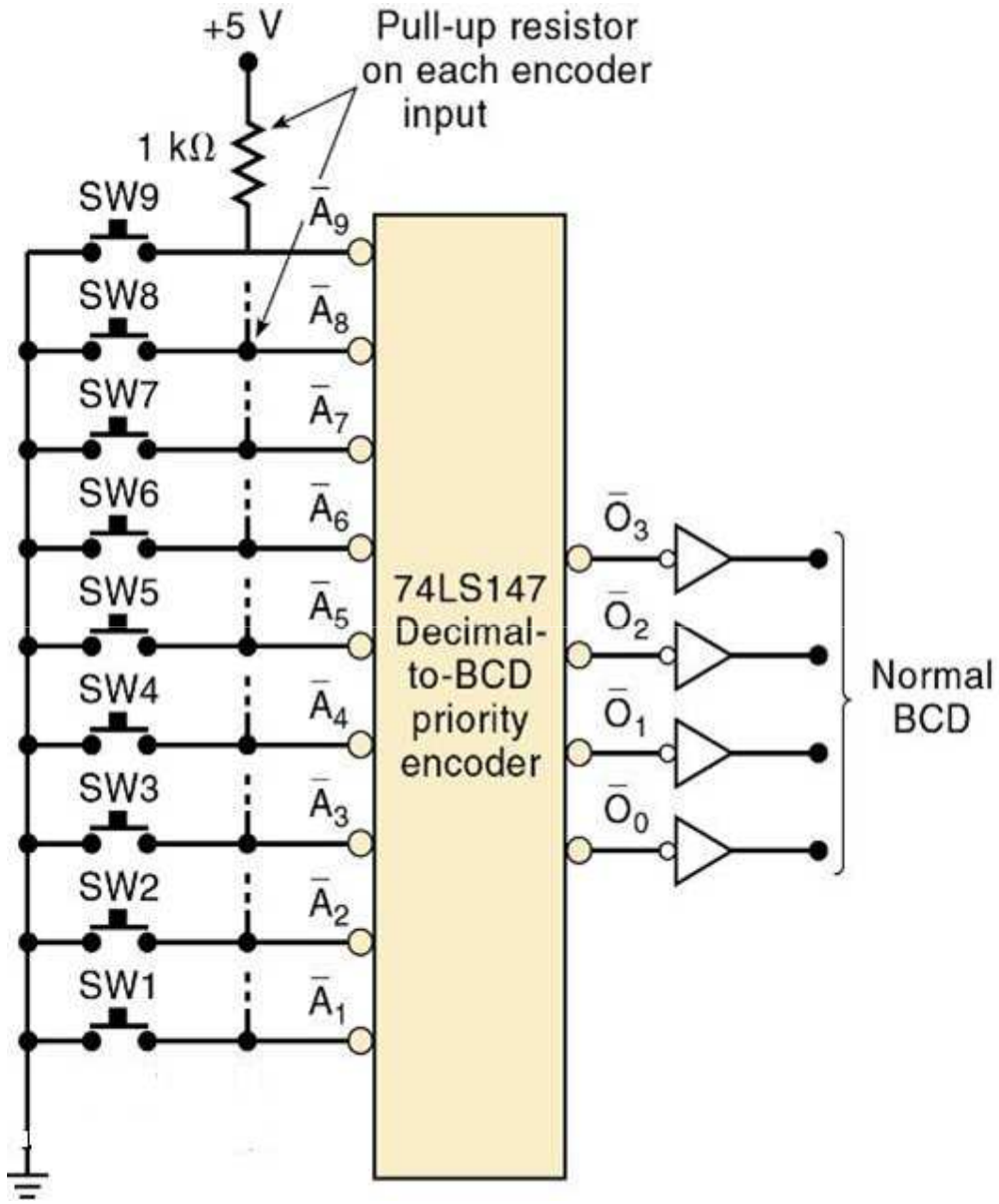
Encoders

The switches are of the normally open type, so the encoder inputs are all normally HIGH.

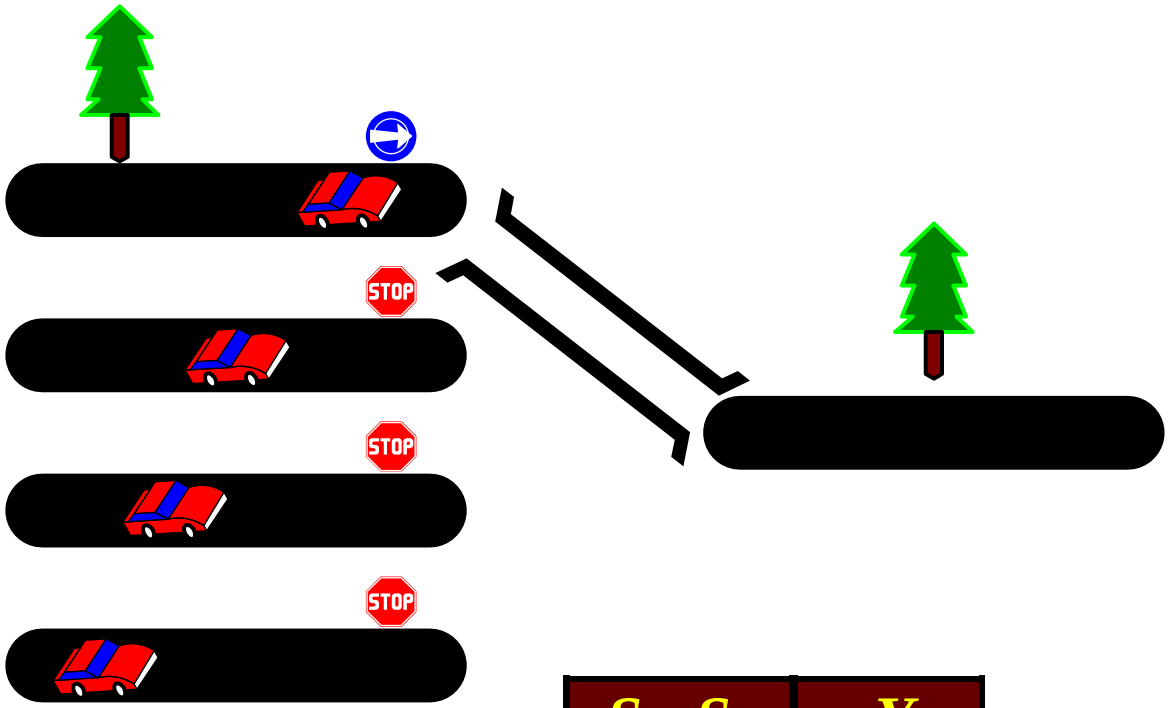
BCD output is 0000.

When a key is depressed, the circuit will produce the BCD code for that digit.

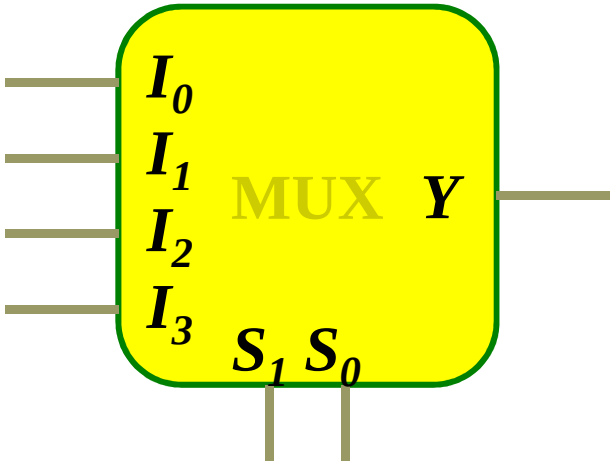
The 74LS147 is a *priority* encoder, so simultaneous key depressions produce the BCD code for the *higher-numbered* key.



Multiplexers

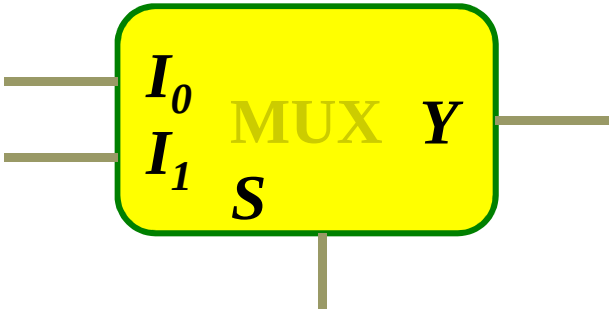


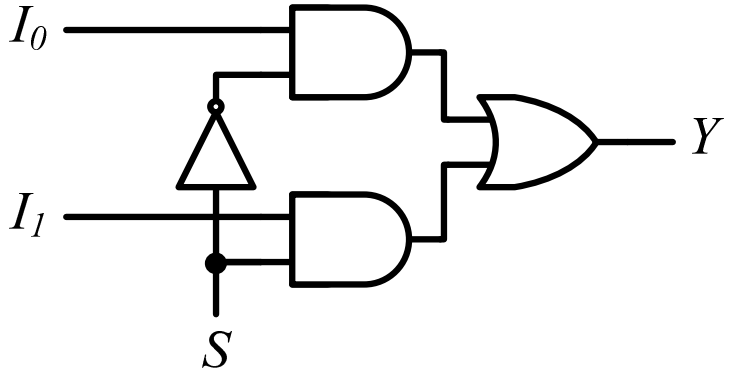
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



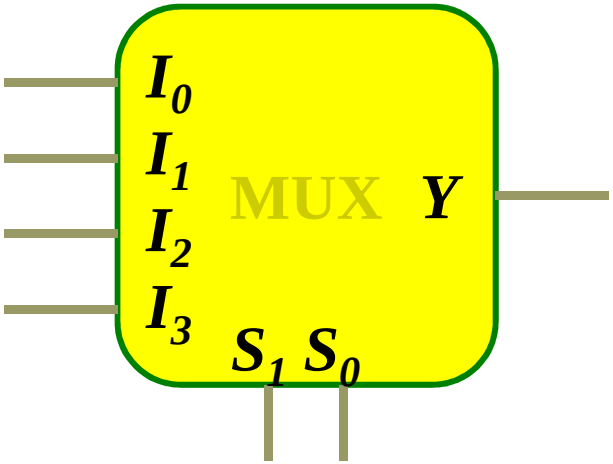
Multiplexers

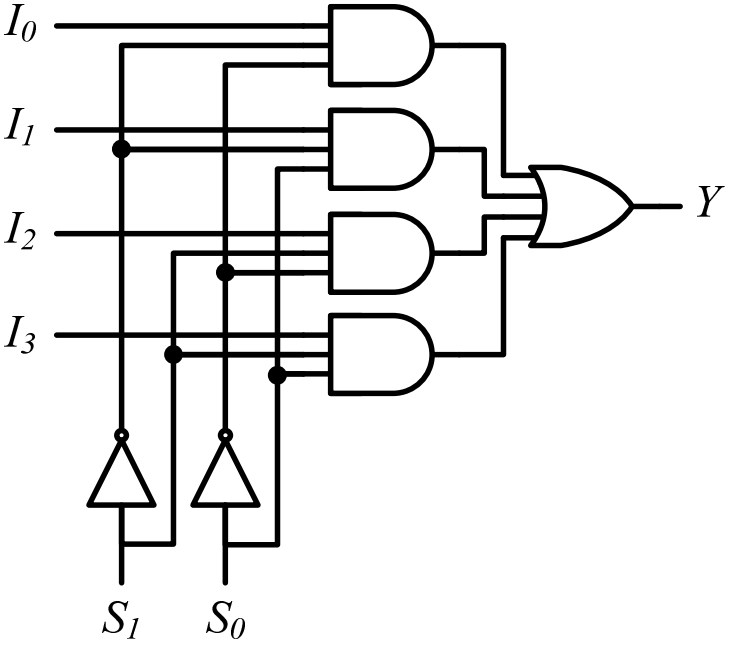
- 2-to-1 MUX





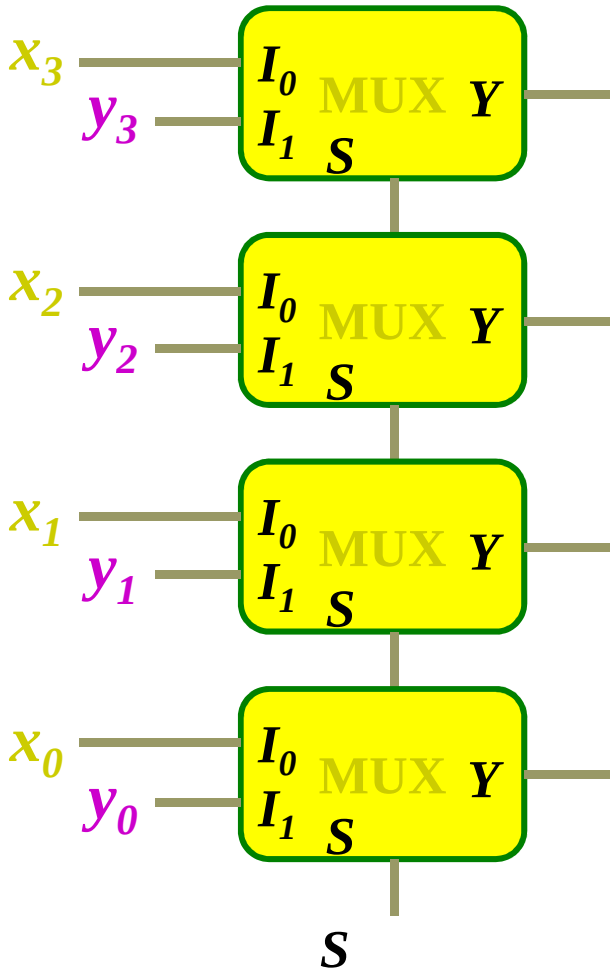
- 4-to-1 MUX

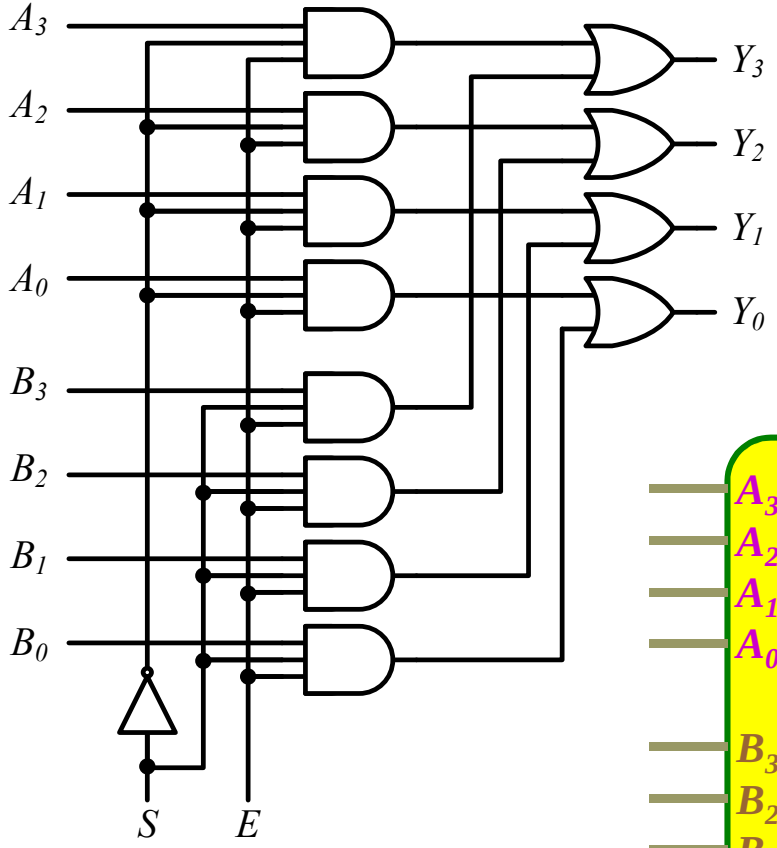


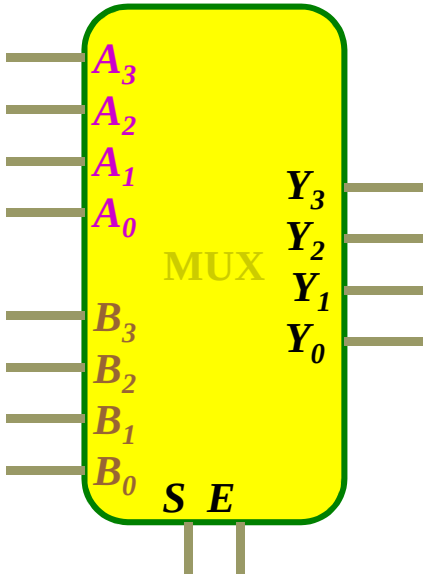


Multiplexers

- Quad 2-to-1 MUX

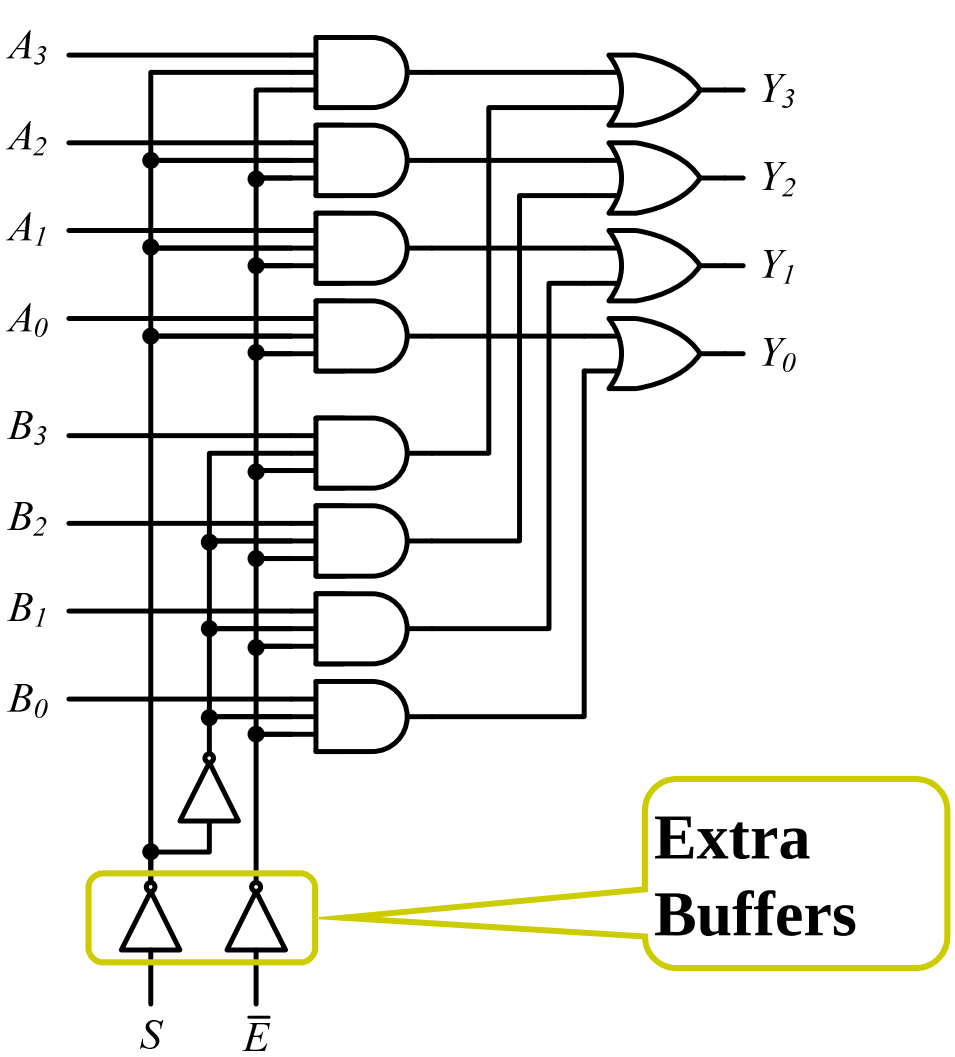


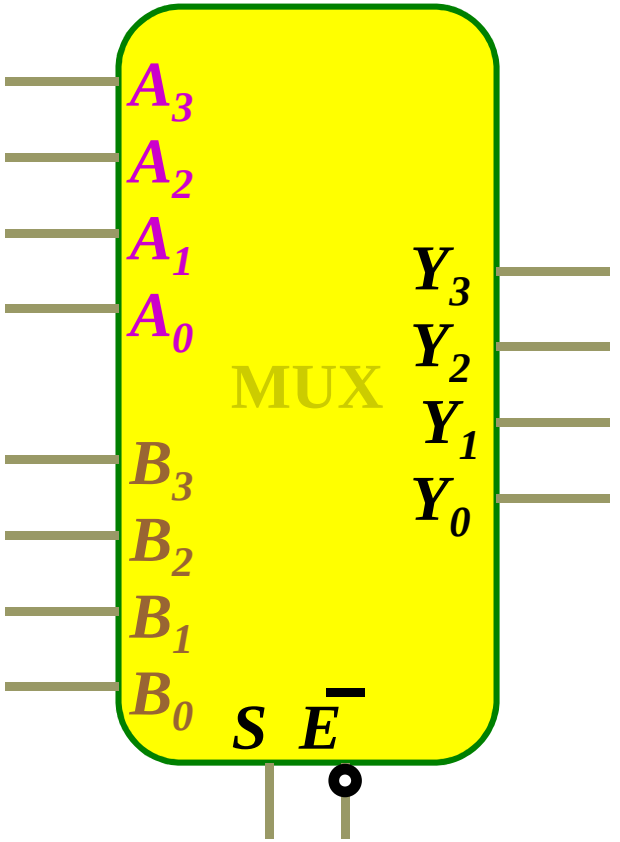




Multiplexers

- Quad 2-to-1 MUX

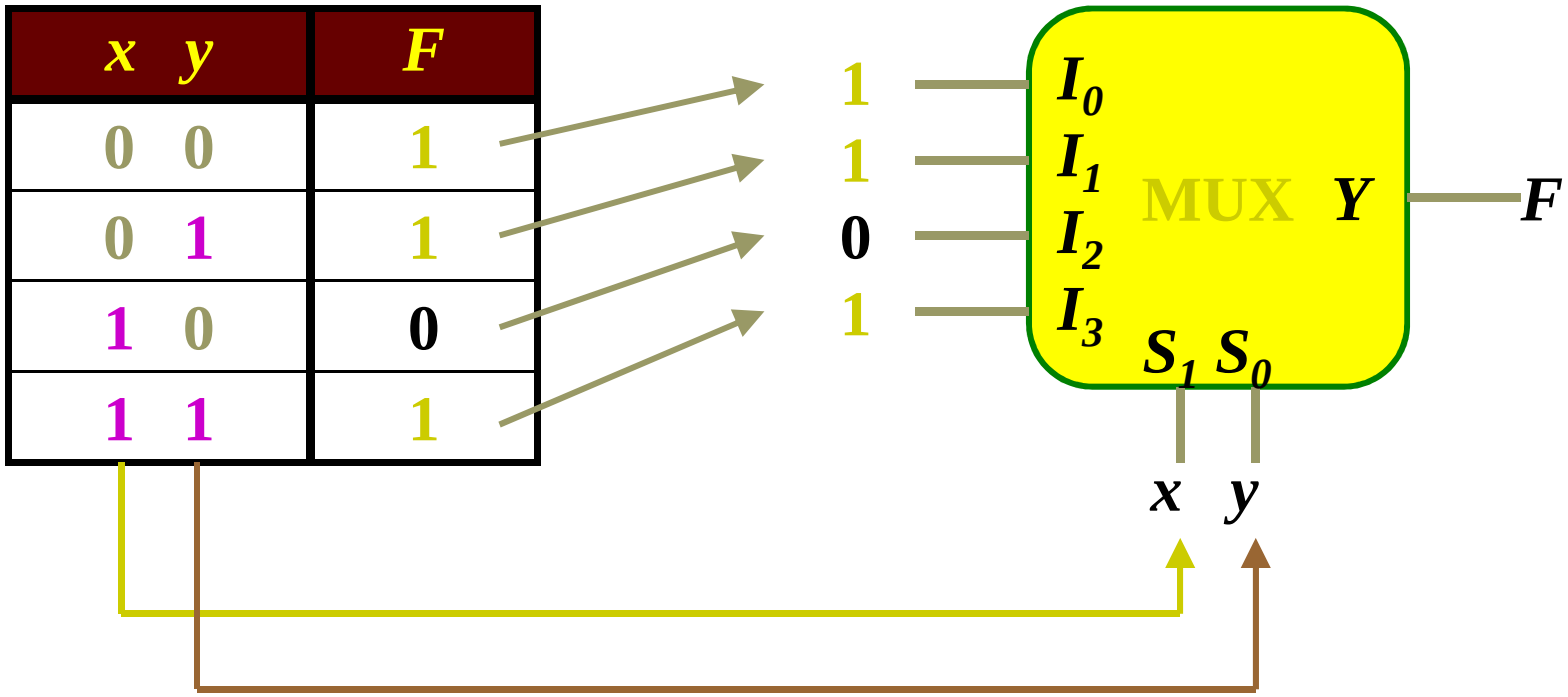




Function Implementation Using Multiplexers

- Example

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

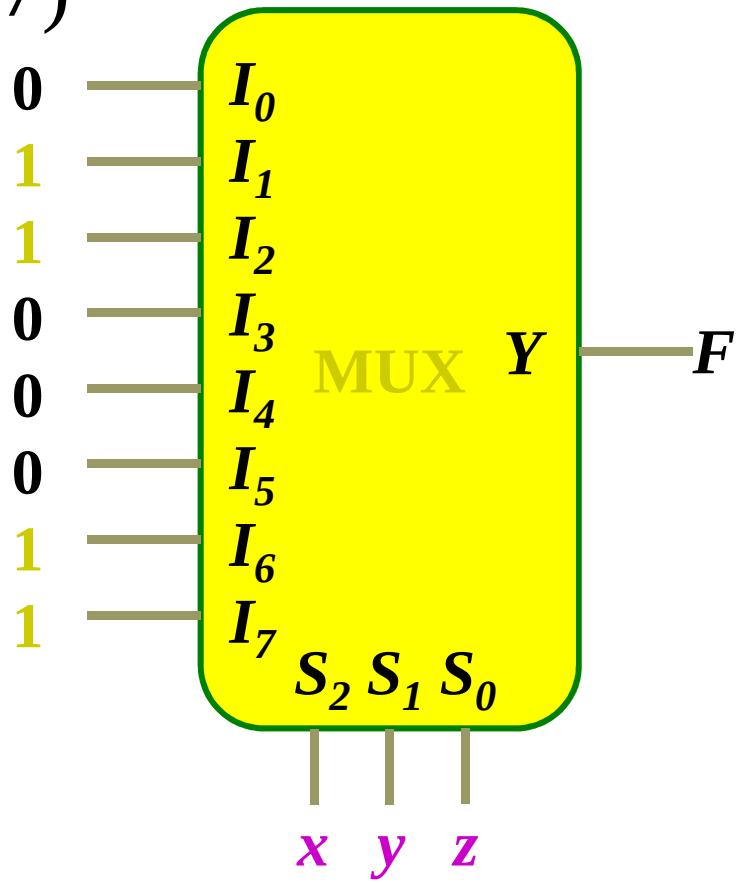


Implementation Using Multiplexers

- 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

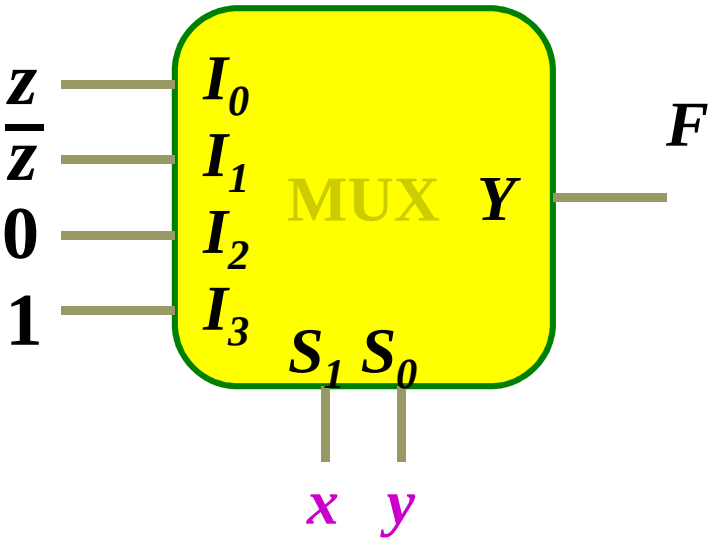


Implementation Using Multiplexers

- Example

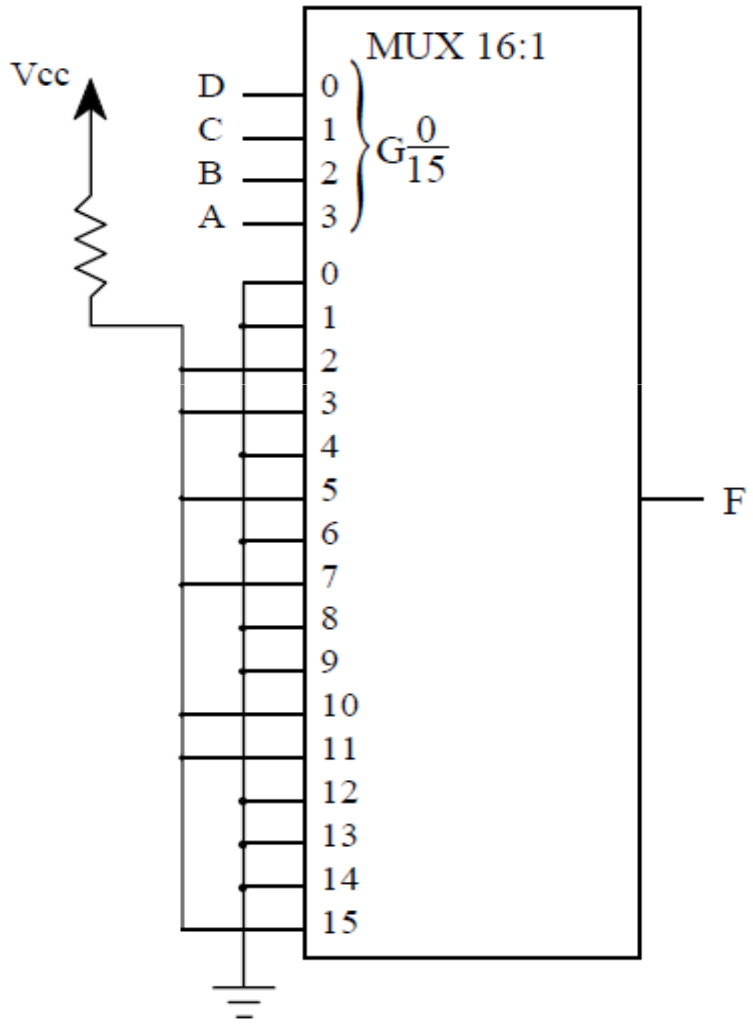
  $F(x, y, z) = \sum(1, 2, 6, 7)$

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



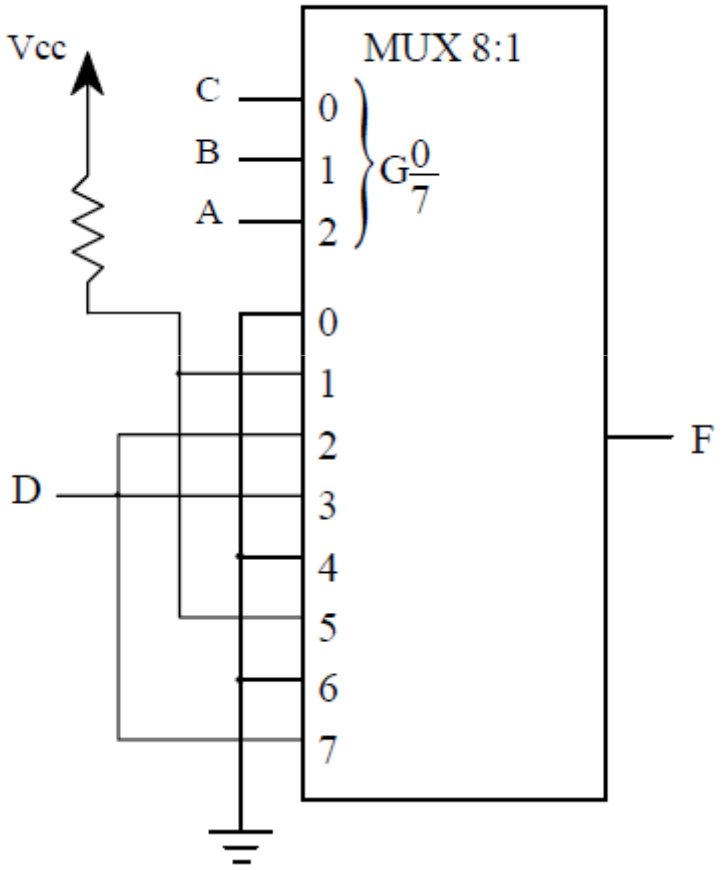
Consider the following truth table that describes a function of 4 Boolean variables.

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



Implementing the above function using 8:1 Mux

A	B	C	D	F	
0	0	0	0	0	0
0	0	0	1	0	
0	0	1	0	1	1
0	0	1	1	1	
0	1	0	0	0	*D*
0	1	0	1	1	
0	1	1	0	0	*D*
0	1	1	1	1	
1	0	0	0	0	0
1	0	0	1	0	
1	0	1	0	1	1
1	0	1	1	1	
1	1	0	0	0	0
1	1	0	1	0	
1	1	1	0	0	*D*
1	1	1	1	1	

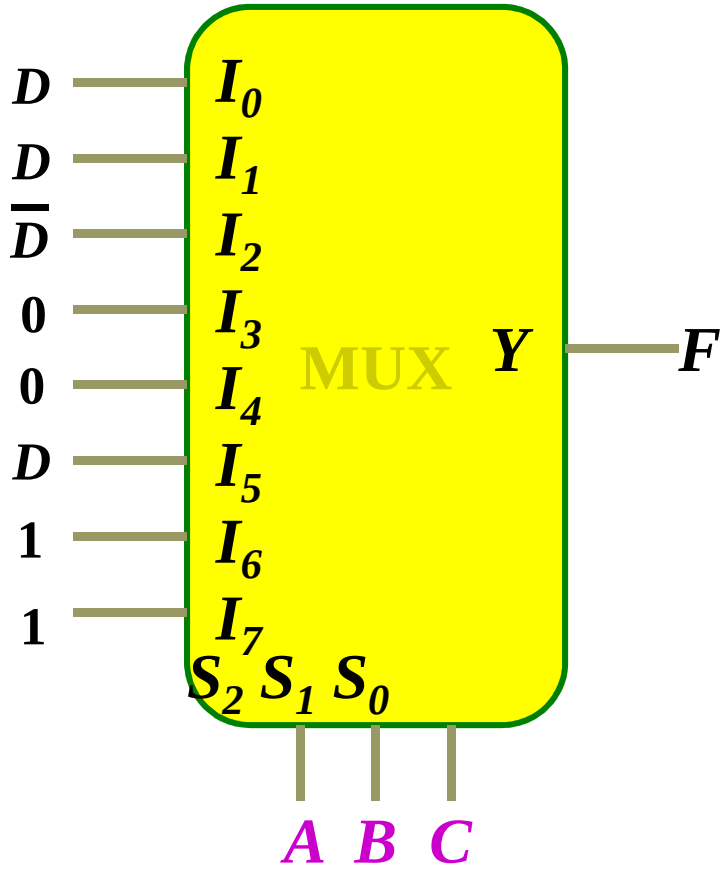


Implementation Using Multiplexers

- Example

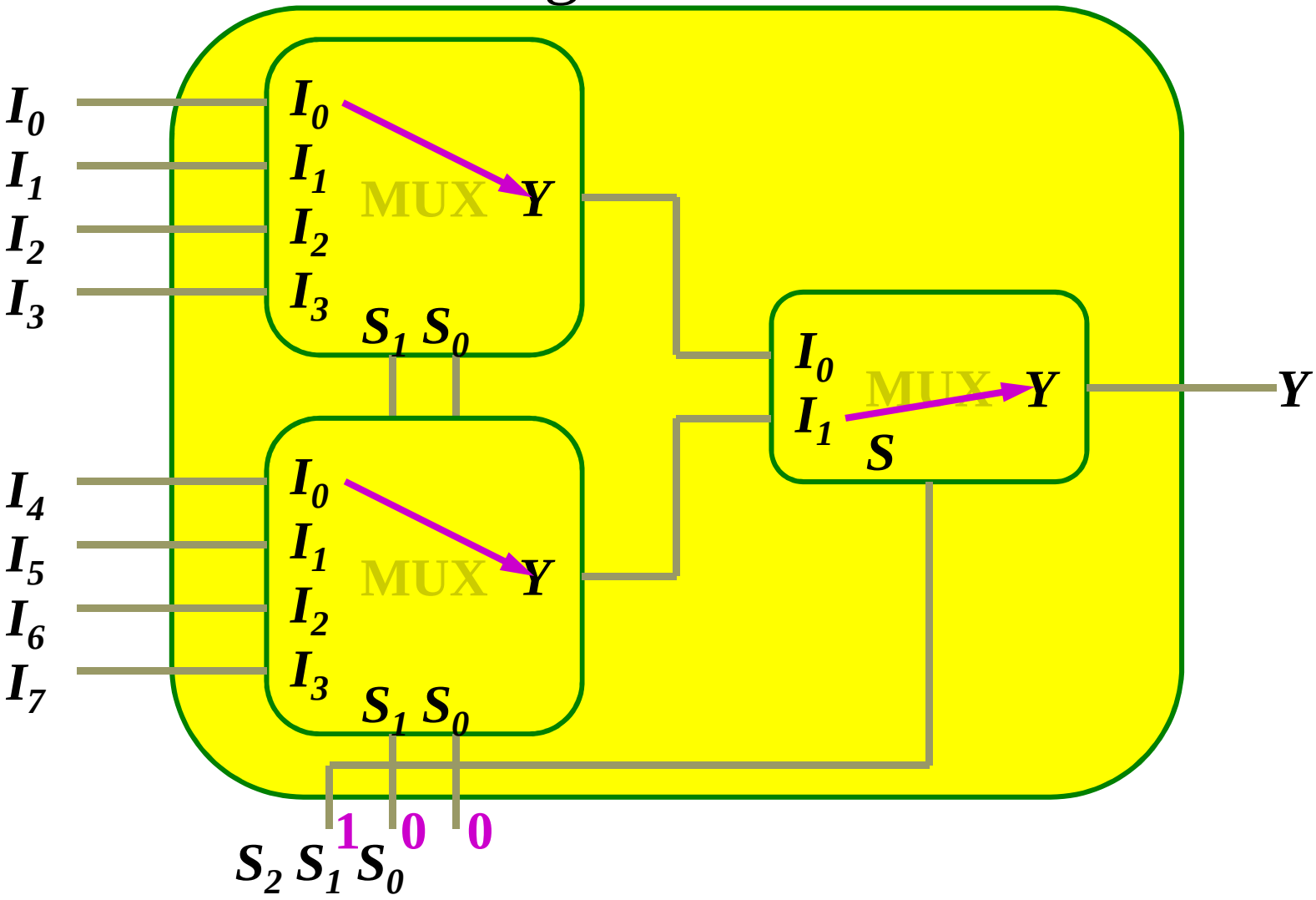
$F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$

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	

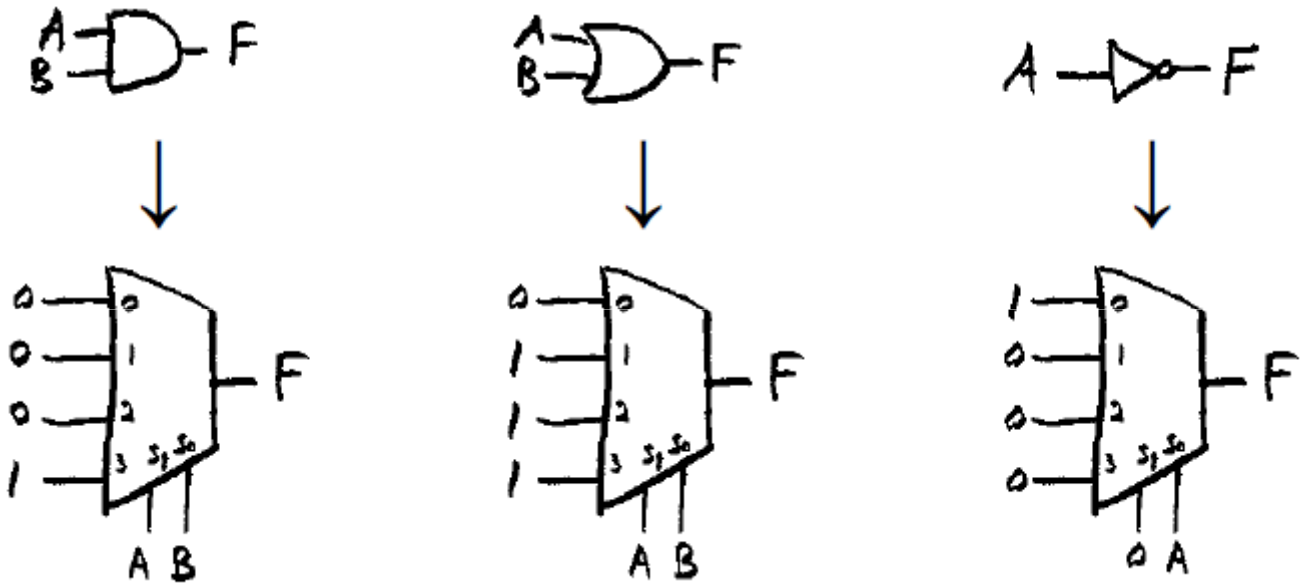


Multiplexer Expansion

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

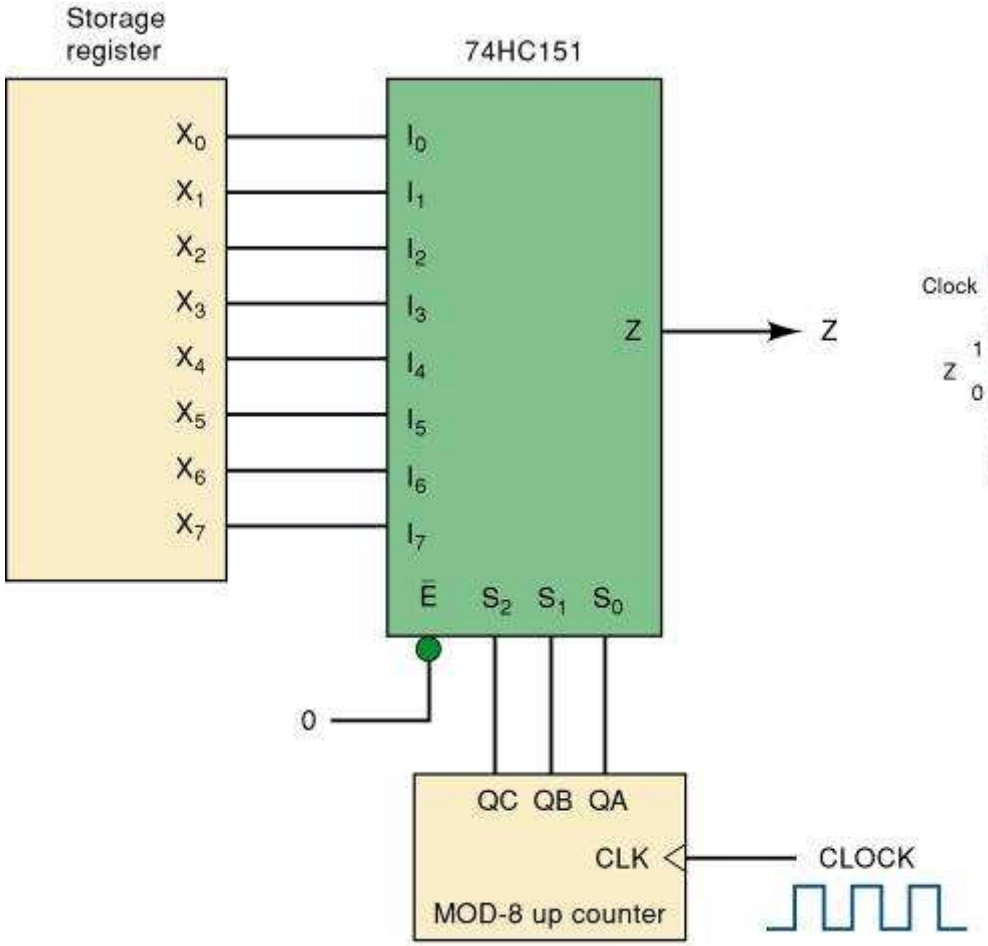


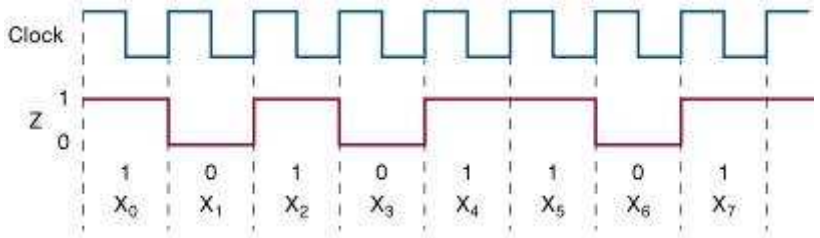
we can use the 4:1 multiplexer to implement these three operations, so it is functionally complete.



Multiplexer Applications

Parallel-to-serial converter.



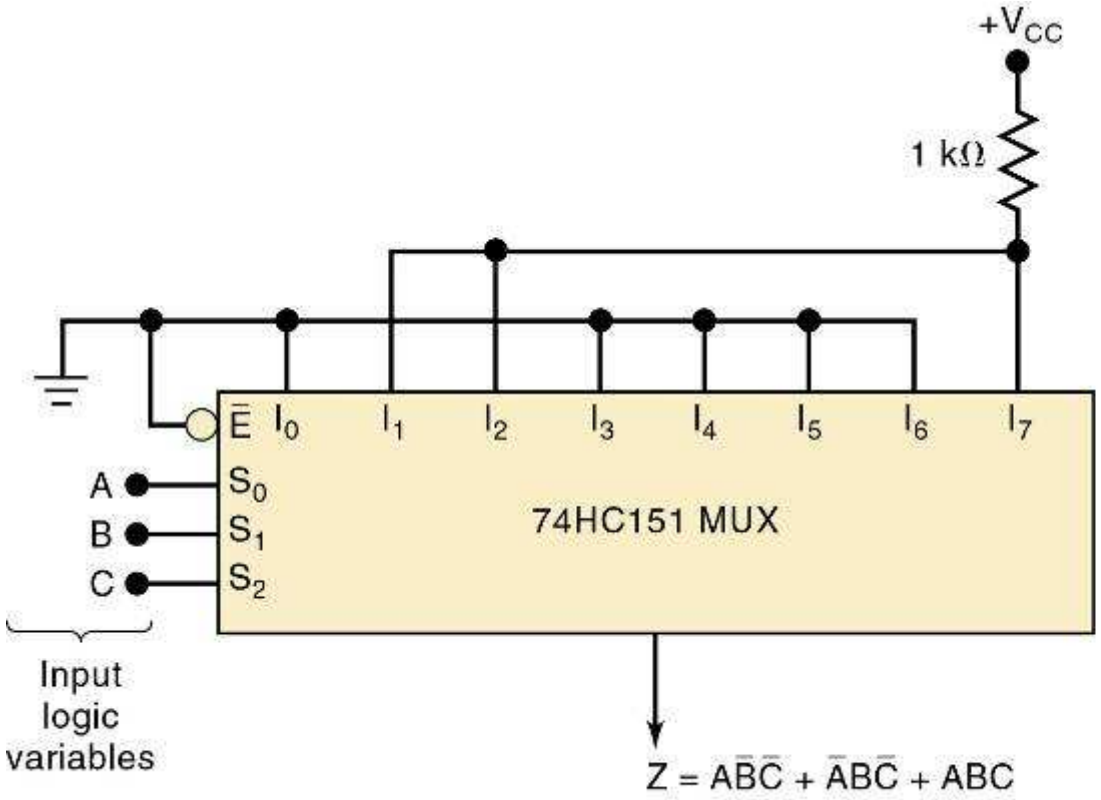


Waveforms for X7X6X5X4X3X2X1X0

10110101

Multiplexer Applications

Multiplexer used to implement a logic function described by the truth table.



C	B	A	Z
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	0
1	1	1	1