# ITI1100 Section Z
# Digital Systems I

**Chapter 4:**

**Combinational Logic (3)**

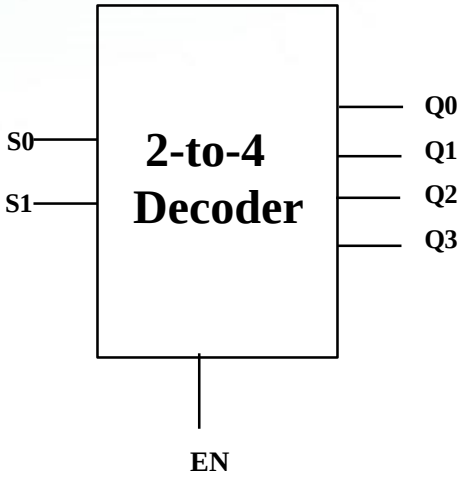
Prof. Mohammad Alja'Afreh

Winter 2017

# Enable Decoder Inputs

- Many devices have an **additional enable input**, which is used to **"activate" or "deactivate" the device**
- For a **decoder**,
  - **EN=1 "activates" the decoder**, so it behaves as specified earlier. **Exactly one of the outputs will be 1**
  - **EN=0 "deactivates" the decoder**. By convention, that means **all of the decoder's outputs are 0**
- We can include this **additional input in the decoder's truth table**

| EN | S1 | S0 | Q0 | Q1 | Q2 | Q3 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |



# Enable Decoder Inputs (cont'd)

- In this table, **whenever EN=0, the outputs are always 0, regardless of inputs S1 and S0**

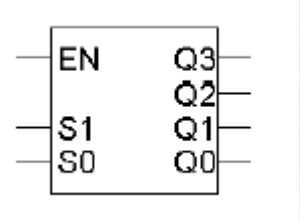
| EN | S1 | S0 | Q0 | Q1 | Q2 | Q3 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

- We can abbreviate the table by writing **x's in the input columns for S1 and S0**

| EN | S1 | S0 | Q0 | Q1 | Q2 | Q3 |
|---|---|---|---|---|---|---|
| 0 | x | x | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

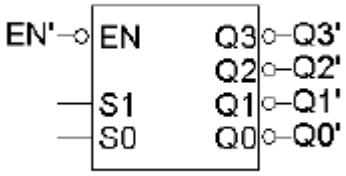
# Active-high and Active-low Decoders

- The decoders studied so far are **active-high decoders** and they are implemented with **AND gates**



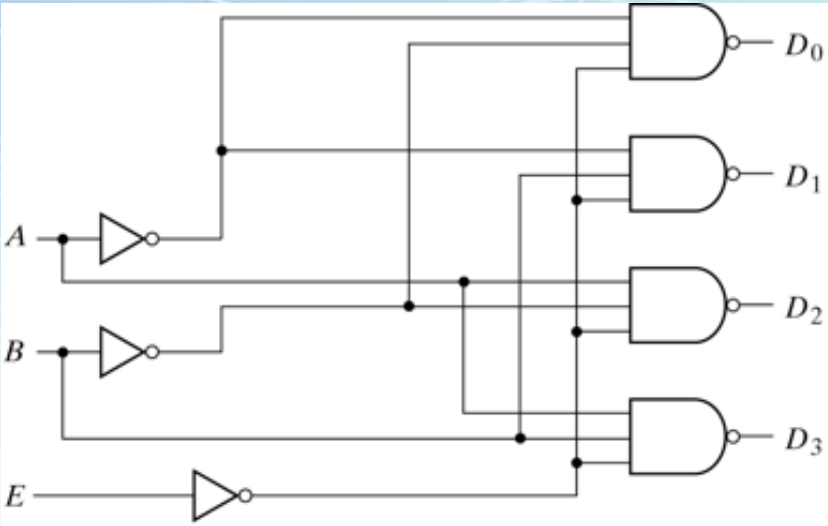

| EN | S1 | S0 | Q0 | Q1 | Q2 | Q3 |
|---|---|---|---|---|---|---|
| 0 | x | x | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

- **Active-low decoders** are implemented using **NAND gates** (i.e. with an inverted EN input and inverted outputs)




| EN | S1 | S0 | Q0 | Q1 | Q2 | Q3 |
|---|---|---|---|---|---|---|
| 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 |
| 1 | x | x | 1 | 1 | 1 | 1 |

# Active-high and Active-low Decoders (cont'd)




**if E = 0**

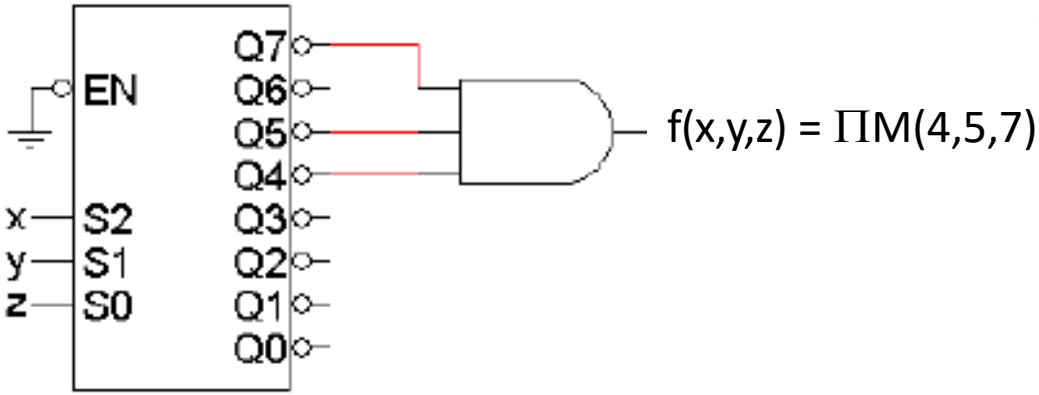
| $E$ | $A$ | $B$ | $D_0$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|---|---|---|
| 1 | $X$ | $X$ | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |

**Active-low 2-to-4 decoder implemented using NAND gates**

# Active-high and Active-low Decoders (cont'd)

- **Active-low decoders with AND gates** can be used to implement **product of maxterms (POS) functions**
- **Example:** f(x,y,z) = $\Pi$M(4,5,7)

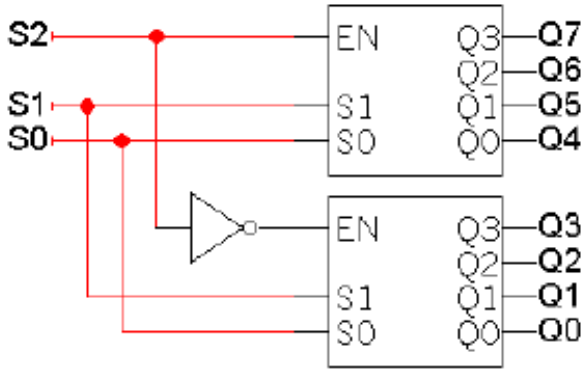
**Since there are 3 inputs and a total of 8 maxterms, we need a 3-to-8 active-low decoder**




**Ground symbol** connected to EN represents logical 0

# Decoder Expansions

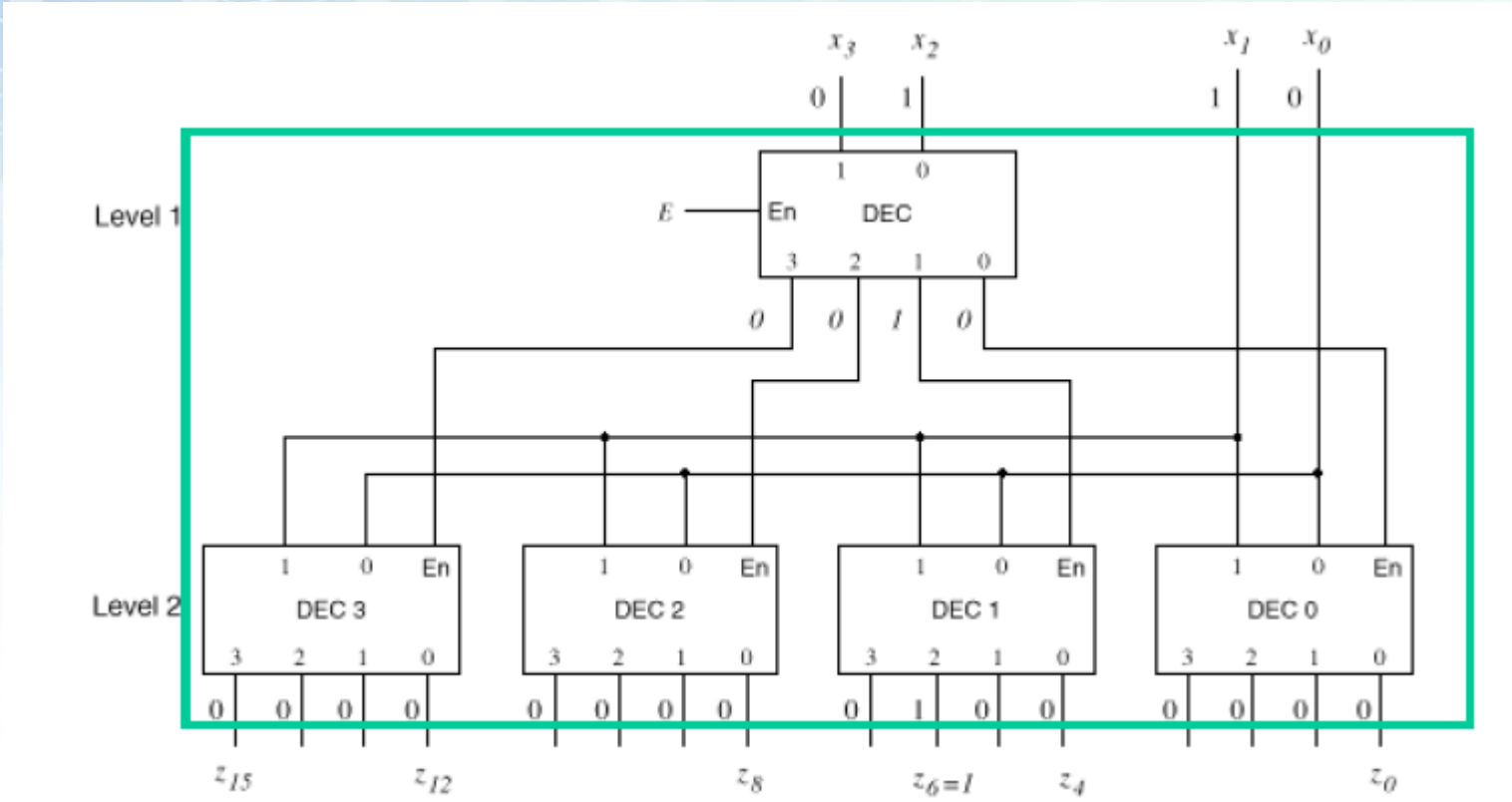
- **Larger decoders** can be **constructed using a number of smaller ones**
- **Enable inputs** are used to **string decoders together**
- **3-to-8 decoder** constructed from **two 2-to-4 decoders**
- **Only one decoder** can be **active at a time**




| S2 | S1 | S0 | Q0 | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

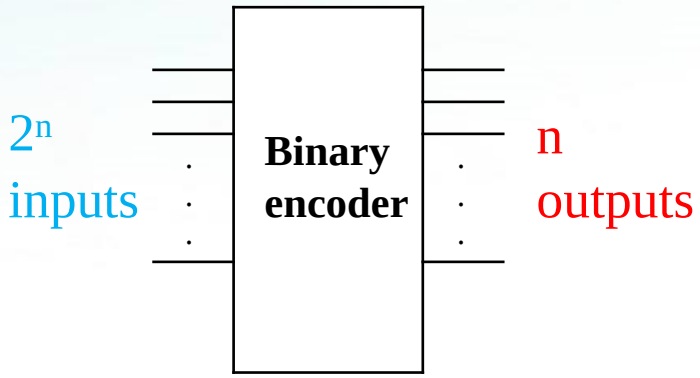
# Decoder Expansions (cont'd)

- A **4-to-16 decoder** can be designed using **five 2-to-4 decoders and one of the 2-to-4 decoders** is used **to generate the enable signals to the four 2-to-4 decoders**



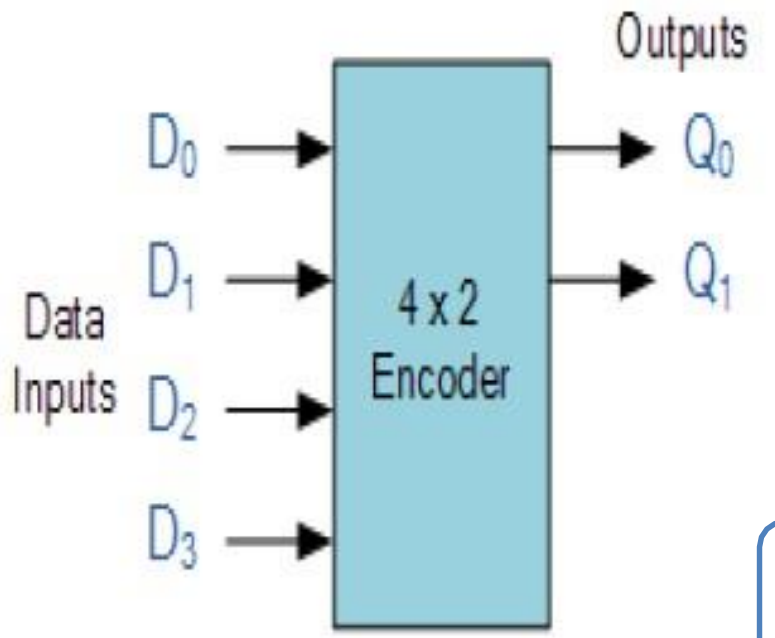
# Encoder

- **The opposite of the decoding process**
- An **"n-bit" binary encoder** has **$2^n$ (or fewer) input lines** and **n-bit output lines**
- **Only one of the input lines is activated at a given time**
- **The output lines generate the binary code corresponding to the input value**



# 4-to-2 Encoder

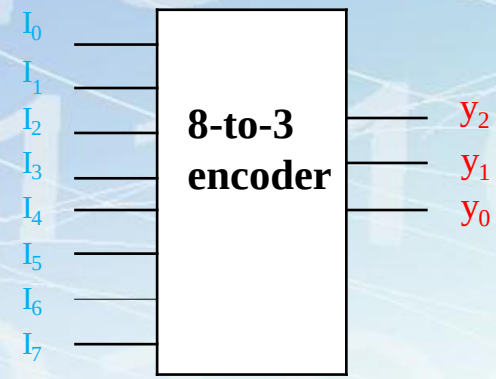
- At any one time, only one input line has a value of 1



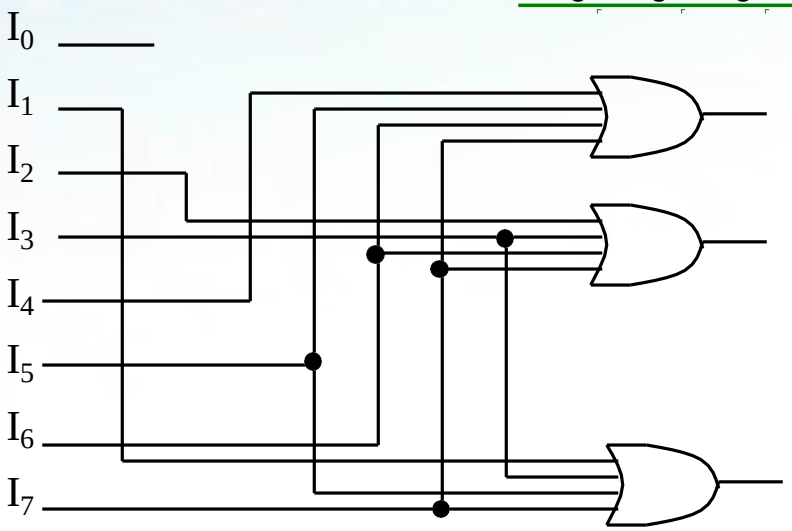

| Inputs | | | | Outputs | |
|---|---|---|---|---|---|
| $D_3$ | $D_2$ | $D_1$ | $D_0$ | $Q_1$ | $Q_0$ |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |

**Truth table in condensed form**

# 8-to-3 Encoder




| Inputs | | | | | | | | Outputs | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $I_0$ | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | $I_6$ | $I_7$ | $y_2$ | $y_1$ | $y_0$ |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |




$y_2 = I_4 + I_5 + I_6 + I_7$

$y_1 = I_2 + I_3 + I_6 + I_7$

$y_0 = I_1 + I_3 + I_5 + I_7$

# Simple Encoder Design Issues

- There are **two ambiguities associated with the design of a simple encoder**:
  - Only one input can be active at any given time. If two inputs are active simultaneously, the output produces an undefined combination (for example, in 8-to-3 encoder **if $I_3$ and $I_4$ are 1 simultaneously, the output of the encoder will be 111 (011 and 100)**
  - An output with all 0's can be generated when **all the inputs are 0's, or when $I_0$ is equal to 1**

# Priority Encoder

- **Solves the ambiguities multiple assigned inputs are allowed; one has priority over all others**
- **Separate output that indicates no input is a 1**

# Priority 4-to-2 Encoder

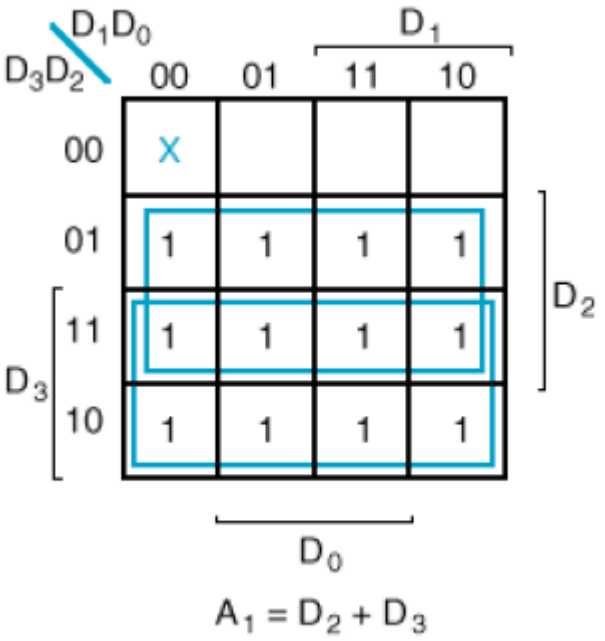
| Inputs | | | | Outputs | | |
|---|---|---|---|---|---|---|
| $D_3$ | $D_2$ | $D_1$ | $D_0$ | $A_1$ | $A_0$ | V |
| 0 | 0 | 0 | 0 | X | X | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | X | 0 | 1 | 1 |
| 0 | 1 | X | X | 1 | 0 | 1 |
| 1 | X | X | X | 1 | 1 | 1 |

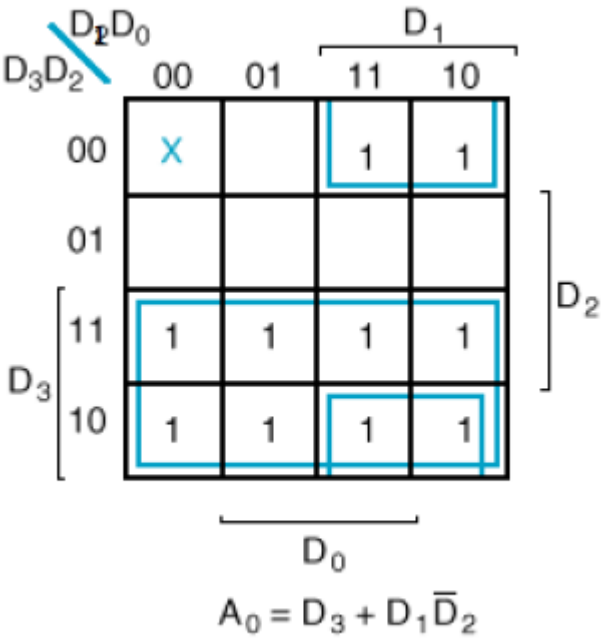
**Truth table in condensed form**

# Priority 4-to-2 Encoder (cont'd)

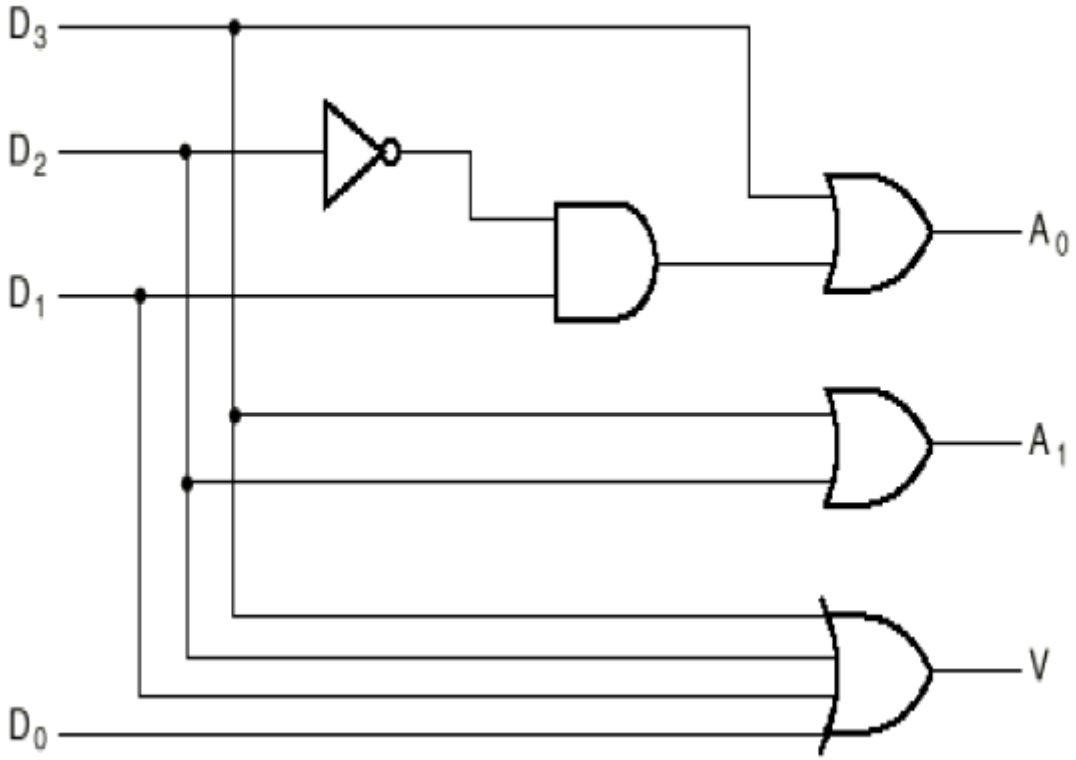
- The operation of the priority encoder is such that:
  - If two or more inputs are equal to 1 at the same time, the input in the **highest-numbered position will take precedence.** $D_3$ has the highest priority
  - A **valid output indicator, designated by V**, is set to 1 only when one or more inputs are equal to 1. $V = D_3 + D_2 + D_1 + D_0$ by inspection

# Priority 4-to-2 Encoder (cont'd)



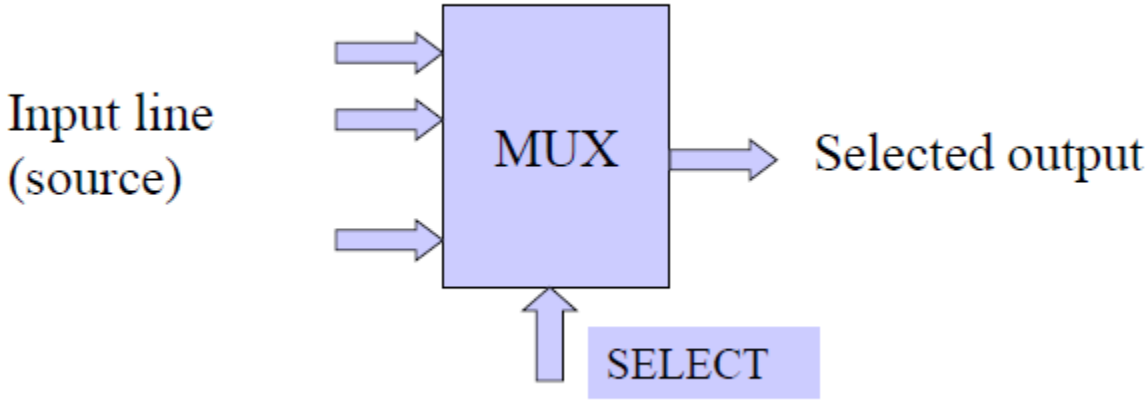
# Priority 4-to-2 Encoder (cont'd)



# (De)Multiplexers and Magnitude Comparators

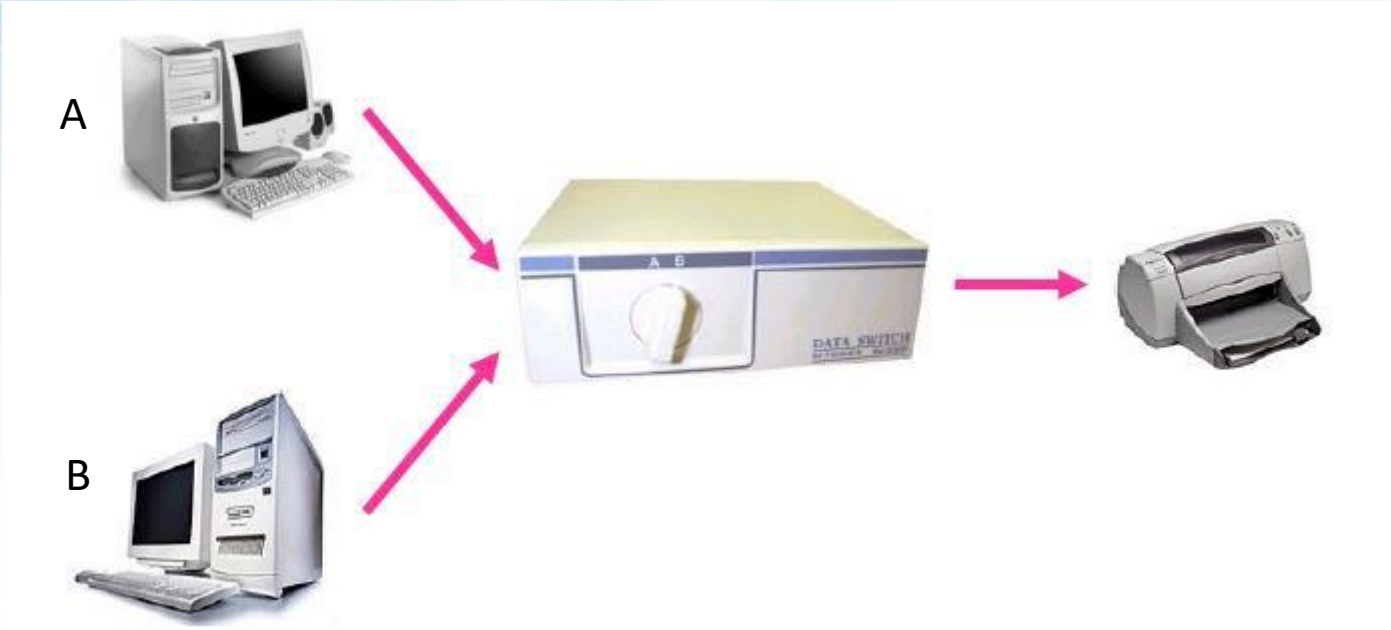
# Multiplexer

- **A multiplexer (MUX) selects one of several input signals and passes it on to the output**
- **Routing of selected data input to the output is controlled by SELECT inputs**



# Multiplexer (cont'd)

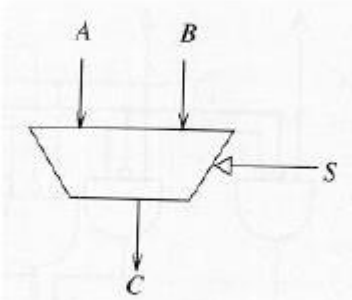
- **A real-life example:** in the old days before networking, **several computers could share one printer** through the use of a manual switch or **MUX**



# Multiplexer (cont'd)

- A combinational circuit with **$2^n$ data inputs**, **1 data output** and **a number of bit (n) control input** that select one of the data inputs

C takes the value of A or B depending on the value of S




2-to-1 Multiplexer

| S | A | B | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

# 2-to-1 Multiplexer

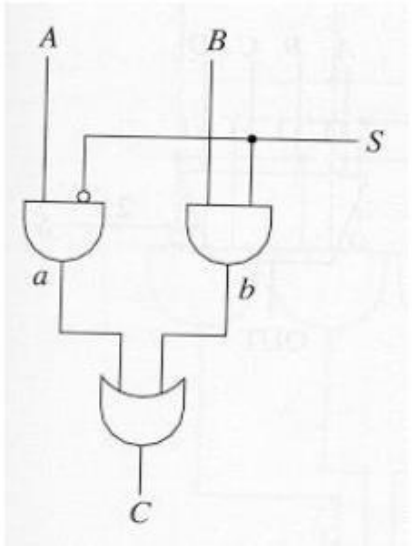
- When S= 0 ⇒ C = A
- When S= 1 ⇒ C = B

Function table

| S | C |
|---|---|
| 0 | A |
| 1 | B |

| Inputs | | | Output |
|---|---|---|---|
| S | A | B | C |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

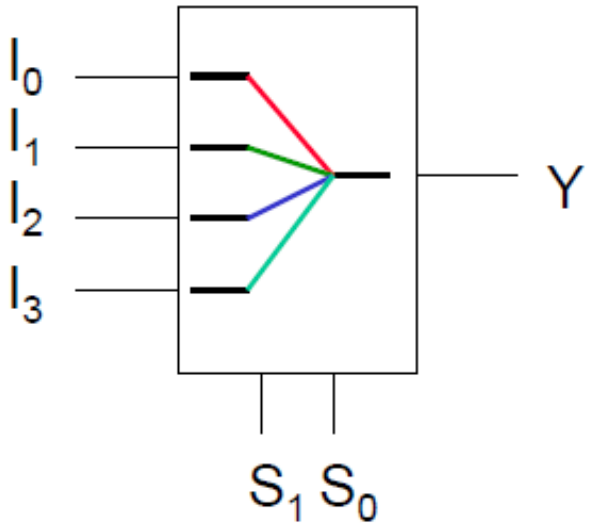
Two level implementation




**C = S'AB' + S'AB + SA'B + SAB = S'A + SB**

# 4-to-1 Multiplexer

| $s_1$ | $s_0$ | $Y$ |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |




if $(S_1S_0)_2 = 0$ Then $Y = I_0$

$$Q = \overline{S_0}.\overline{S_1}.I_o$$

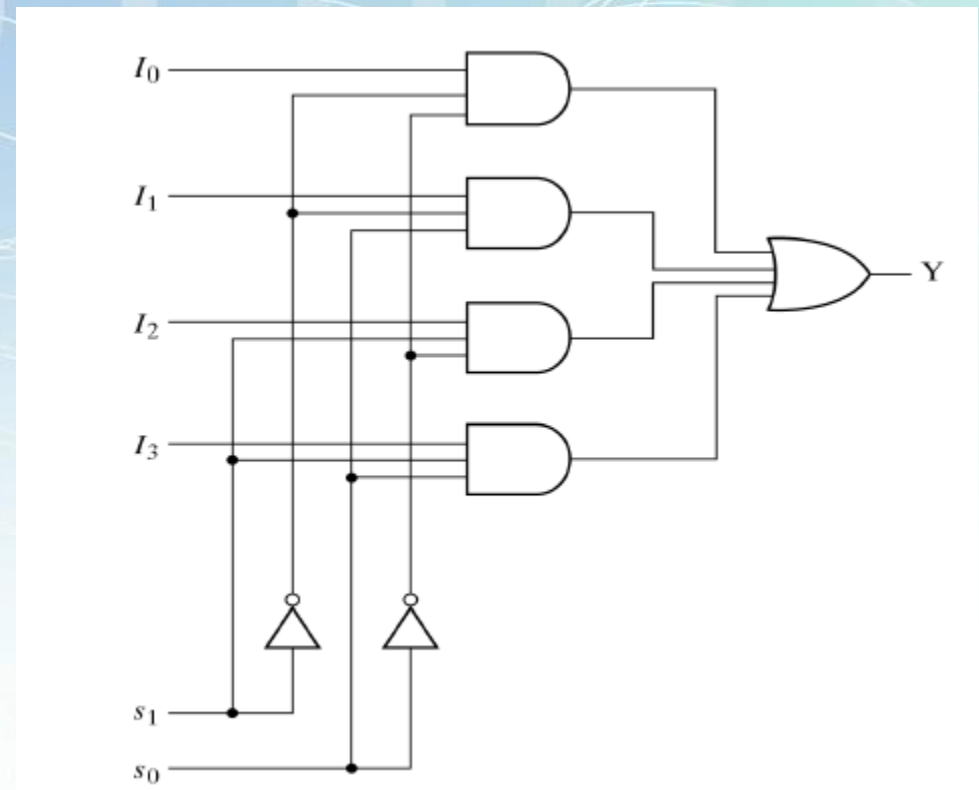
if $(S_1S_0)_2 = 1$ Then $Y = I_1$

$$Q = S_0.\overline{S_1}.I_1$$

and so on, thus we have

$$Y = \overline{S_1}.\overline{S_0}.I_0 + \overline{S_1}.S_0.I_1 + S_1.\overline{S_0}.I_2 + S_1.S_0.I_3$$

# 4-to-1 Multiplexer (cont'd)




$$Y = \overline{S_1}.\overline{S_0}.I_0 + \overline{S_1}.S_0.I_1 + S_1.\overline{S_0}.I_2 + S_1.S_0.I_3$$

# Implementing Functions with Multiplexers

- **Multiplexers** can be used to **implement Boolean functions**. **One way** to implement **a function of n variables** is to use an **$2^n$-to-1 MUX**:
  - For **each minterm** of the function, **logic 1 is connected to data input Di**. Each data input corresponds to one row of the truth table
  - **Connect the function's input variables to select inputs**. These are used to indicate a particular input combination
- **Example:** f(x,y,z) = Σm(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 |

**8x1 MUX**