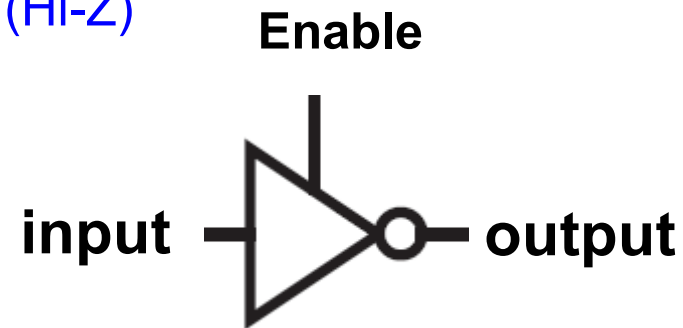


**Building Blocks:**  
**Tri-State Buffer**  
**Decoder**  
**Encoder**

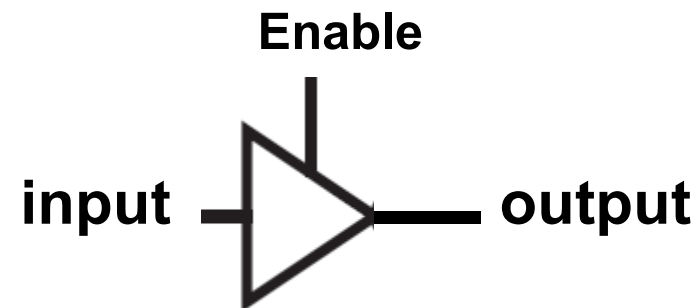
# Tri-State (Output Enable) Gate

- **Tri-State (Three-State) Inverter:**

- The output is the NOT of input if the Enable input is HIGH
- Else output is Hi-Impedance (Hi-Z)
  - Unconnected

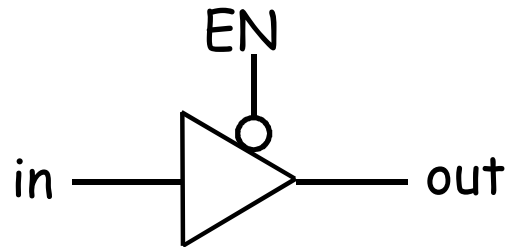


- **Tri-State Buffer:**



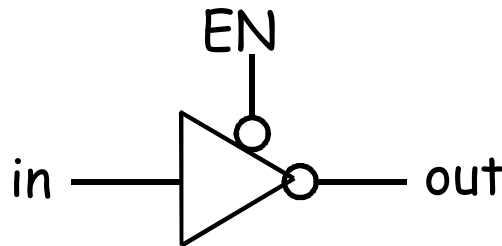
# Active Low Input

- **Tri-State Buffer with Active Low Enable:**



Tri-state BUF, EN low

- **Tri-State Inverter with Active Low Enable:**

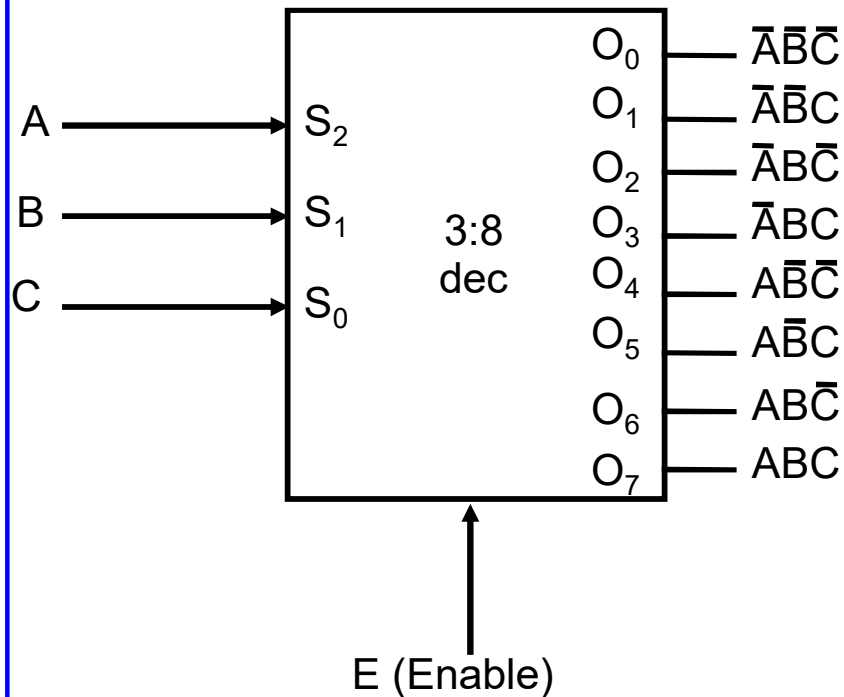


Tri-state INV, EN low

# Decoder Description

- **Binary decoder is a very common combinational circuit used as a building block for:**
  - Implementing general logic
  - Making larger circuits such as Read-Only Memories (ROM)
  - Steering logic in large designs (also called glue logic)
  - Can be found in  $n$ -to- $2^n$  denominations
    - e.g., 2-to-4, 3-to-8, etc.
  - For simplicity, will be called “decoder”

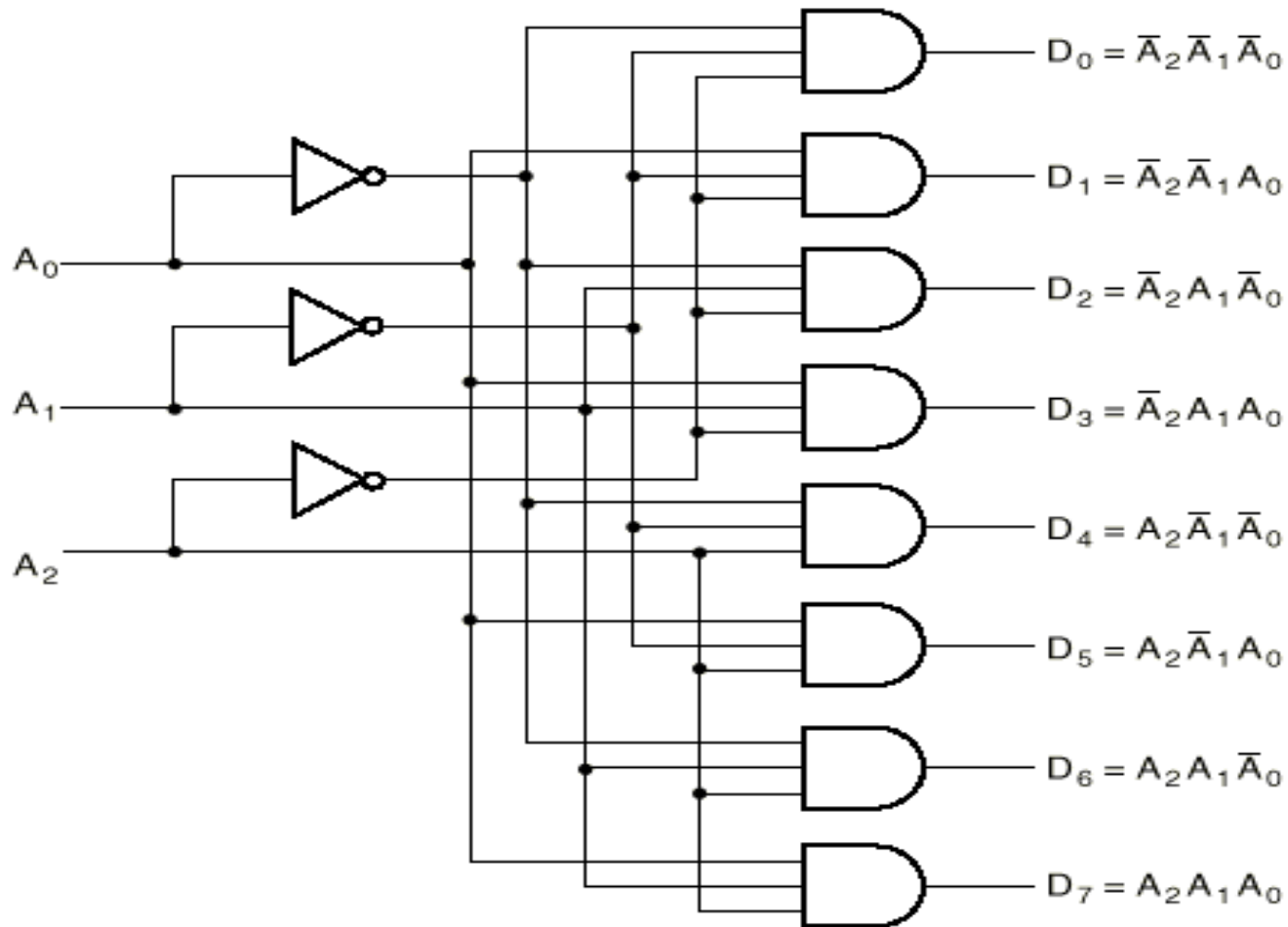
# A 3-to-8 Decoder



E	A	B	C	$O_0$	$O_1$	$O_2$	$O_3$	$O_4$	$O_5$	$O_6$	$O_7$
0	X	X	X	0	0	0	0	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	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

- Each decoder output is a minterm:
- Any combinational circuit can be constructed using decoders and OR gates

# Decoder Internal Circuit



# Design Using Decoder

- **Basic Idea:**

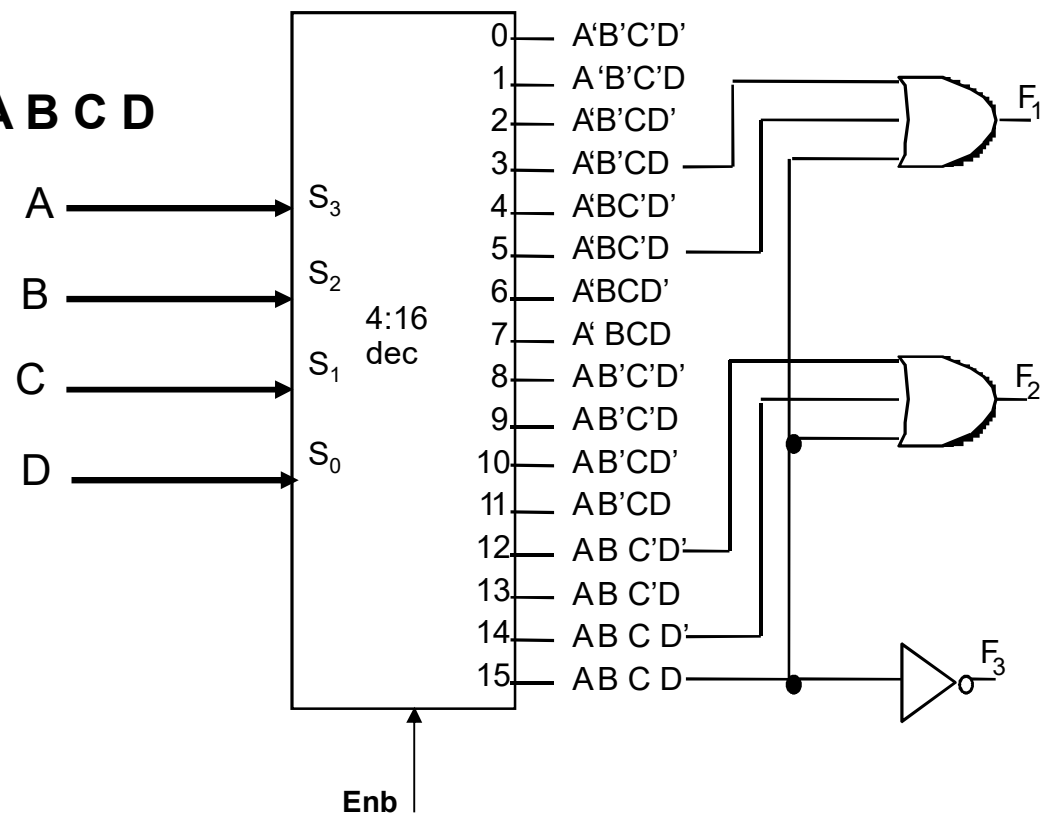
- A decoder can be used to implement any combinational circuit (reminder: each decoder output is a minterm)

- **Example:**

$$F1 = A' B C' D + A' B' C D + A B C D$$

$$F2 = A B C' D' + A B C$$

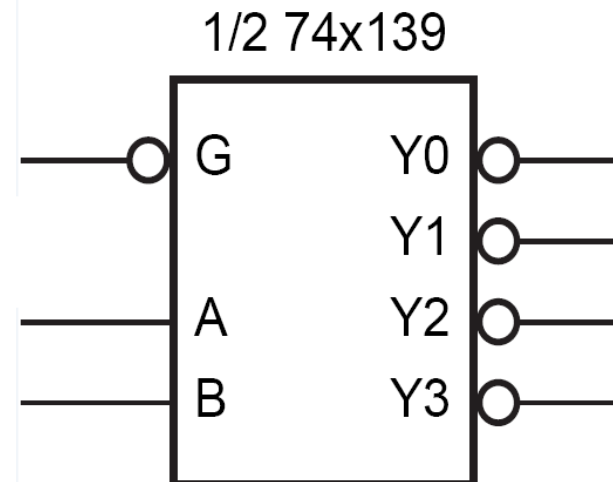
$$F3 = A' + B' + C' + D'$$



# Inputs/Outputs Can Be Active Low

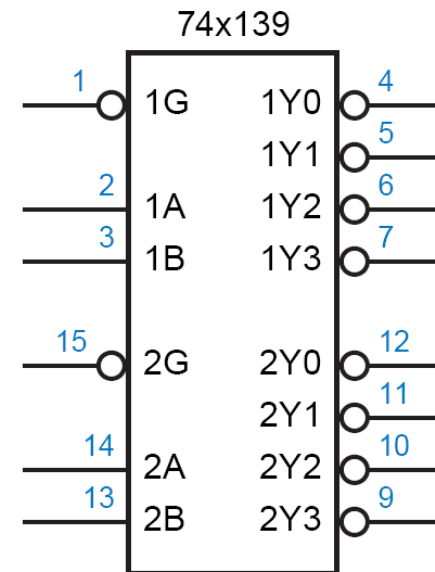
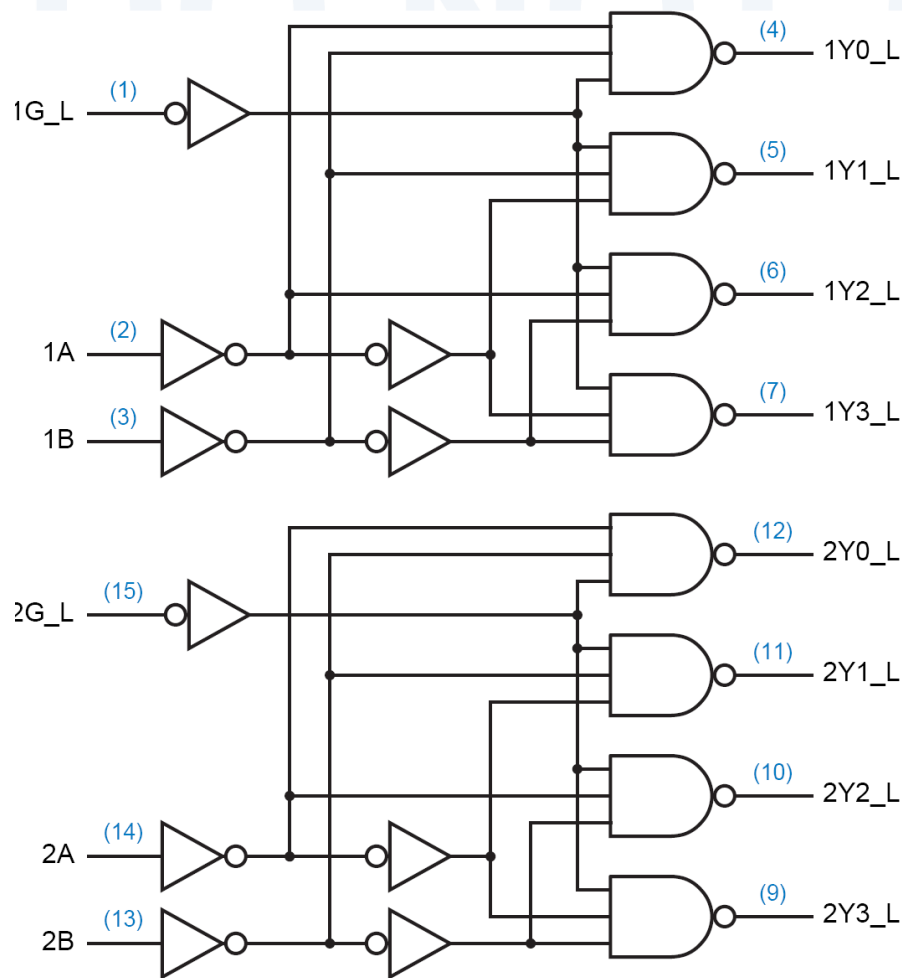
- **Decoder with**
  - Active Low Enable
  - Active Low Outputs

G	A	B	Y <sub>0</sub>	Y <sub>1</sub>	Y <sub>2</sub>	Y <sub>3</sub>
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



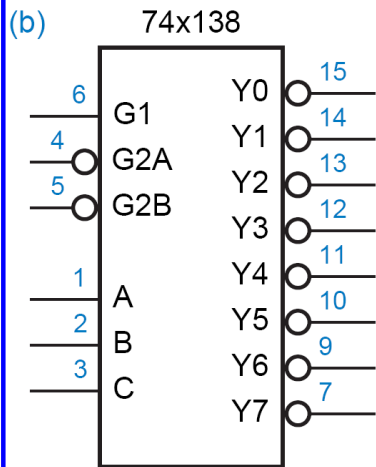


# 74x139 dual 2-to-4 decoder



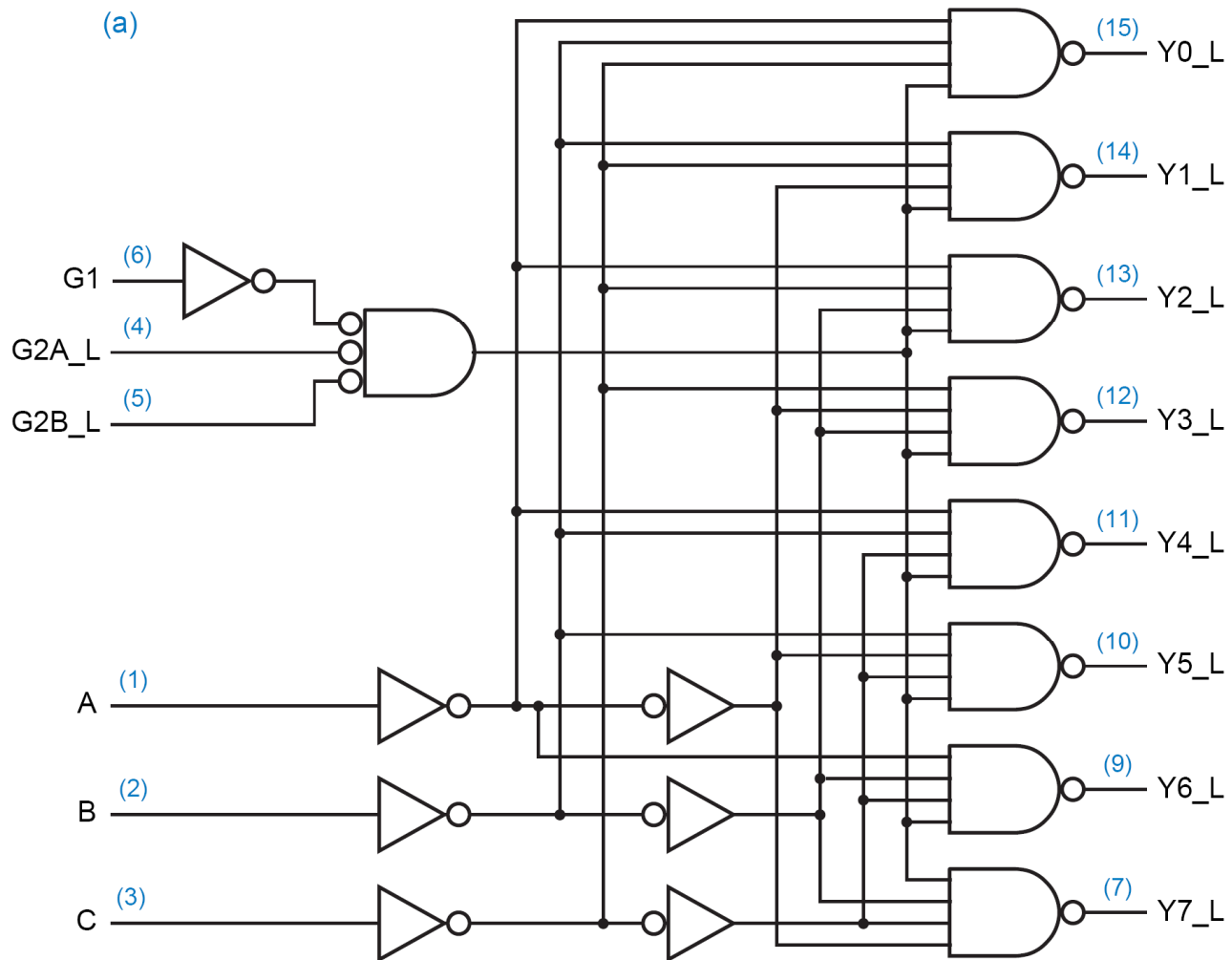
Inputs			Outputs			
G_L	B	A	Y3_L	Y2_L	Y1_L	Y0_L
1	x	x	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

# 74x138 3-8 Decoder



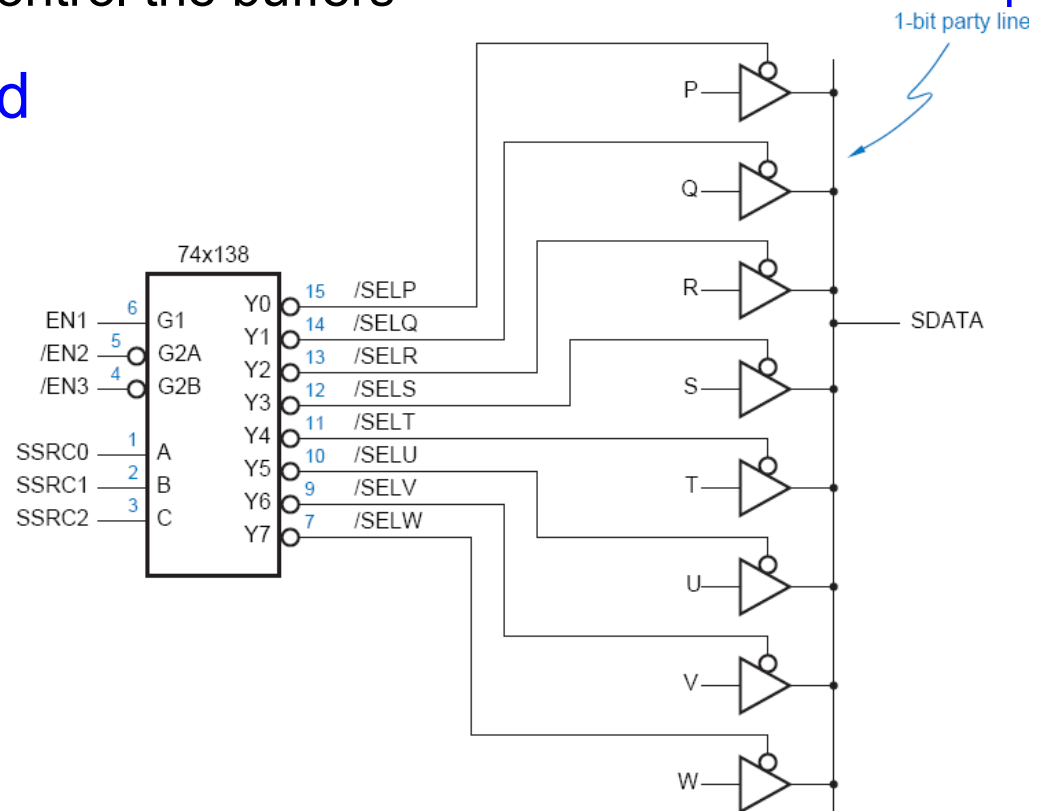
Inputs						Outputs							
G1	G2A_L	G2B_L	C	B	A	Y7_L	Y6_L	Y5_L	Y4_L	Y3_L	Y2_L	Y1_L	Y0_L
0	x	x	x	x	x	1	1	1	1	1	1	1	1
x	1	x	x	x	x	1	1	1	1	1	1	1	1
x	x	1	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1

# 74x138 3-8 Decoder



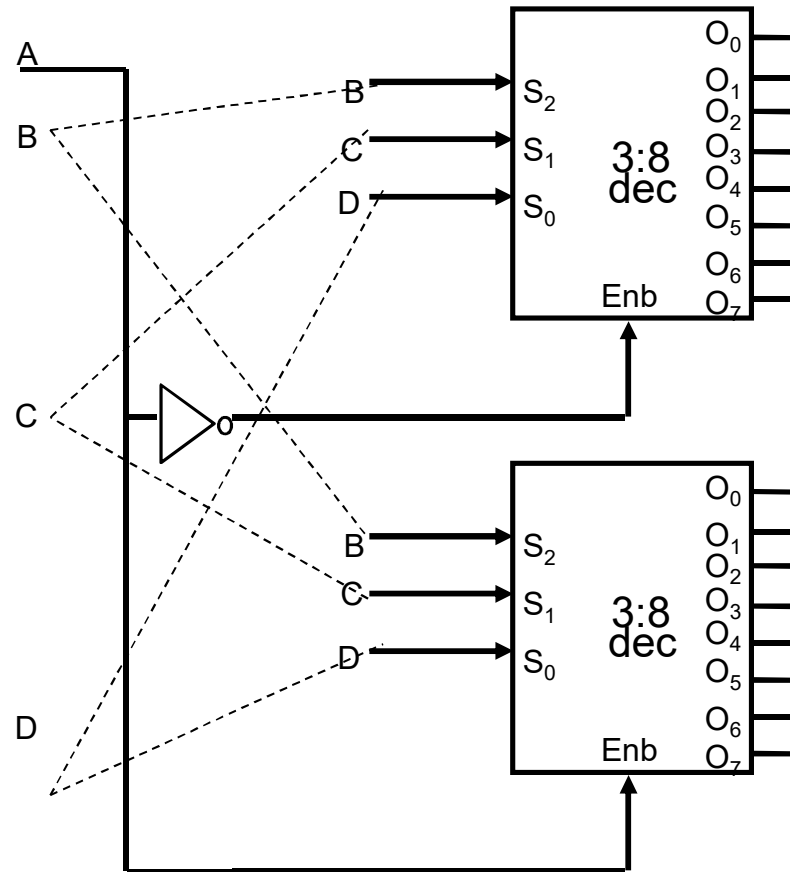
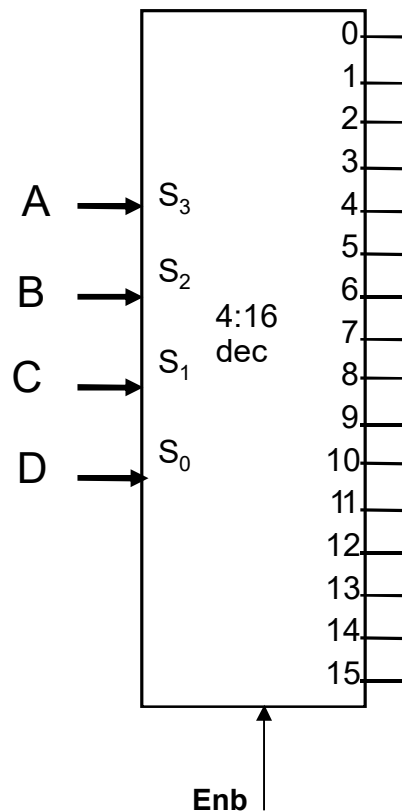
# Application: Device Selection

- Normally, gate outputs cannot be tied together
  - Have to use tri-state buffers to do that
  - Have to make sure the buffers are not on simultaneously
  - Can use a decoder to control the buffers
- Example: a decoder and 8 tri-state buffers are used to share a single line among 8 devices
- The decoder guarantees that no two buffers are on simultaneously
- Some decoders have built-in tri-state outputs



# Larger Decoders?

- Can build a decoder by using smaller decoders
- Example: a 4-to-16 based on two 3-to-8 decoders



# Decoders

- How to build a 5-32 decoder by using 4-16 and 2-4 decoders?

# Decoders

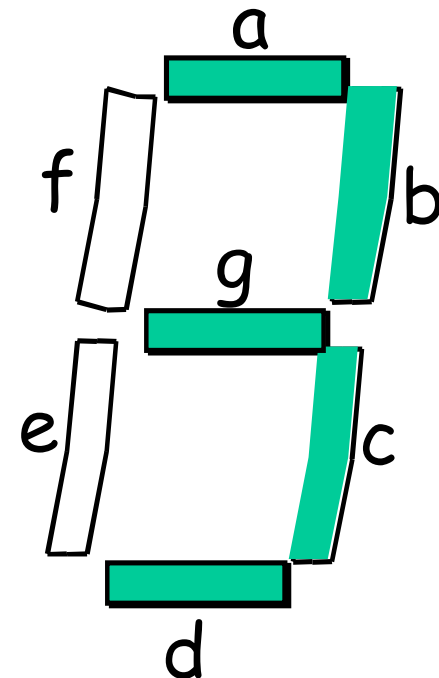
- **Decoder: a more general term**
  - Our focus was on “binary decoders”

# 7-Segment Decoder



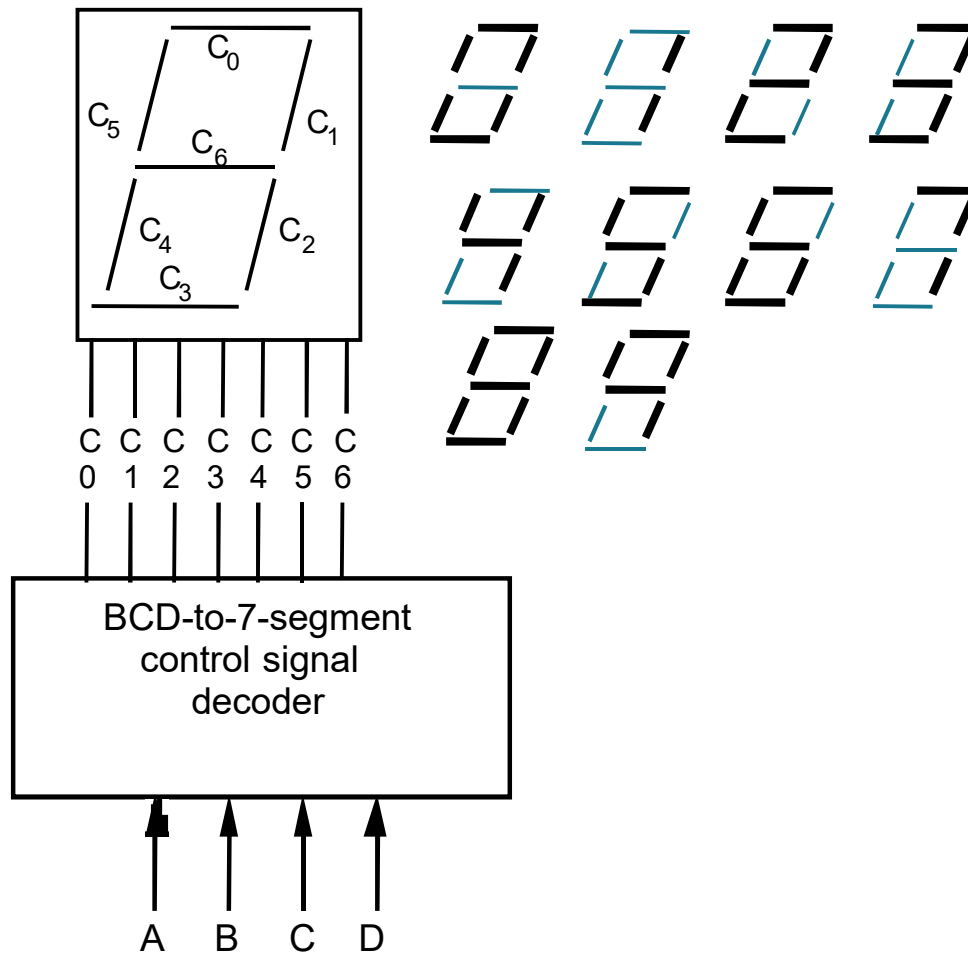
- **Seven-segment display:**

- 7 LEDs (light emitting diodes), each one controlled by an input
- 1 means “on”, 0 means “off”
- Display digit “3”?
  - Set a, b, c, d, g to 1
  - Set e, f to 0





# 7-Segment Decoder



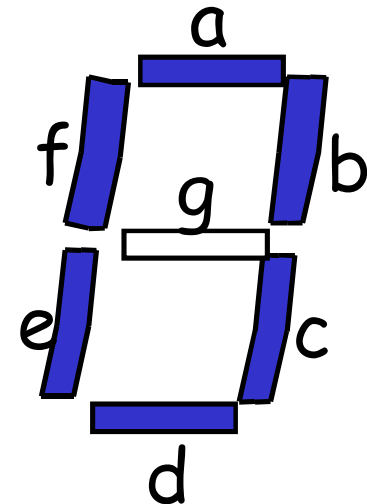
# 7-Segment Decoder

- **7-Segment Decoder:**

- Input is a 4-bit BCD code  $\rightarrow$  4 inputs (A, B, C, D).
- Output is a 7-bit code (a,b,c,d,e,f,g) that allows for the decimal equivalent to be displayed.

- **Example:**

- Input:  $0000_{\text{BCD}}$
- Output:  $1111110$   
(a=b=c=d=e=f=1, g=0)



# BCD-to-7Segment Truth Table

Digit	ABCD	abcdefg
0	0000	1111110
1	0001	0110000
2	0010	1101101
3	0011	1111001
4	0100	0110011
5	0101	1011011
6	0110	X011111
7	0111	11100X0

Digit	ABCD	abcdefg
8	1000	1111111
9	1001	111X011
	1010	XXXXXXX
	1011	XXXXXXX
	1100	XXXXXXX
	1101	XXXXXXX
	1110	XXXXXXX
	1111	XXXXXXX

??

# K-maps

AB \ CD		A			
		00	01	11	10
C	00	1	0	X	1
	01	0	1	X	1
	11	1	1	X	X
	10	1	1	X	X

K-map for a

AB \ CD		A			
		00	01	11	10
C	00	1	1	X	1
	01	1	0	X	1
	11	1	1	X	X
	10	1	0	X	X

K-map for b

AB \ CD		A			
		00	01	11	10
C	00	1	1	X	1
	01	1	1	X	1
	11	1	1	X	X
	10	0	1	X	X

K-map for c

AB \ CD		A			
		00	01	11	10
C	00	1	0	X	1
	01	0	1	X	0
	11	1	0	X	X
	10	1	1	X	X

K-map for d

AB \ CD		A			
		00	01	11	10
C	00	1	0	X	1
	01	0	0	X	0
	11	0	0	X	X
	10	1	1	X	X

K-map for e

AB \ CD		A			
		00	01	11	10
C	00	1	1	X	1
	01	0	1	X	1
	11	0	0	X	X
	10	0	1	X	X

K-map for f

AB \ CD		A			
		00	01	11	10
C	00	0	1	X	1
	01	0	1	X	1
	11	1	0	X	X
	10	1	1	X	X

K-map for g

$$\begin{aligned} a &= A + B D + C + B' D' \\ b &= A + C' D' + C D + B' \\ c &= A + B + C' + D \end{aligned}$$

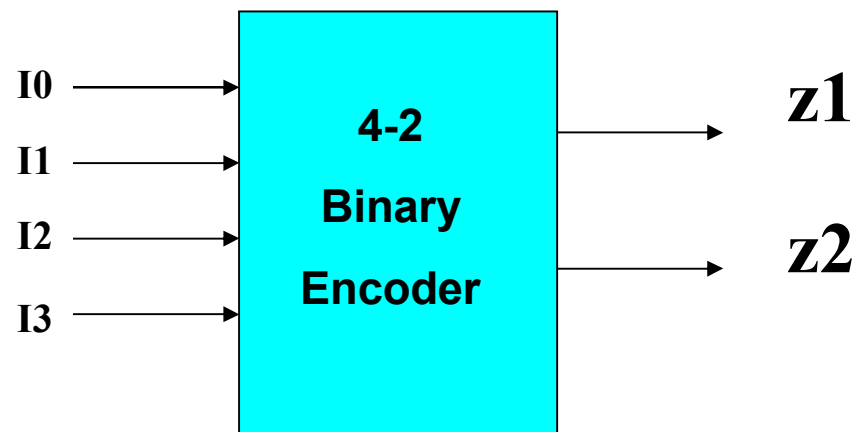
$$\begin{aligned} d &= B' D' + C D' + B C' D + B' C \\ e &= B' D' + C D \\ f &= A + C' D' + B D' + B C' \\ g &= A + C D' + B C' + B' C \end{aligned}$$

**Encoder**

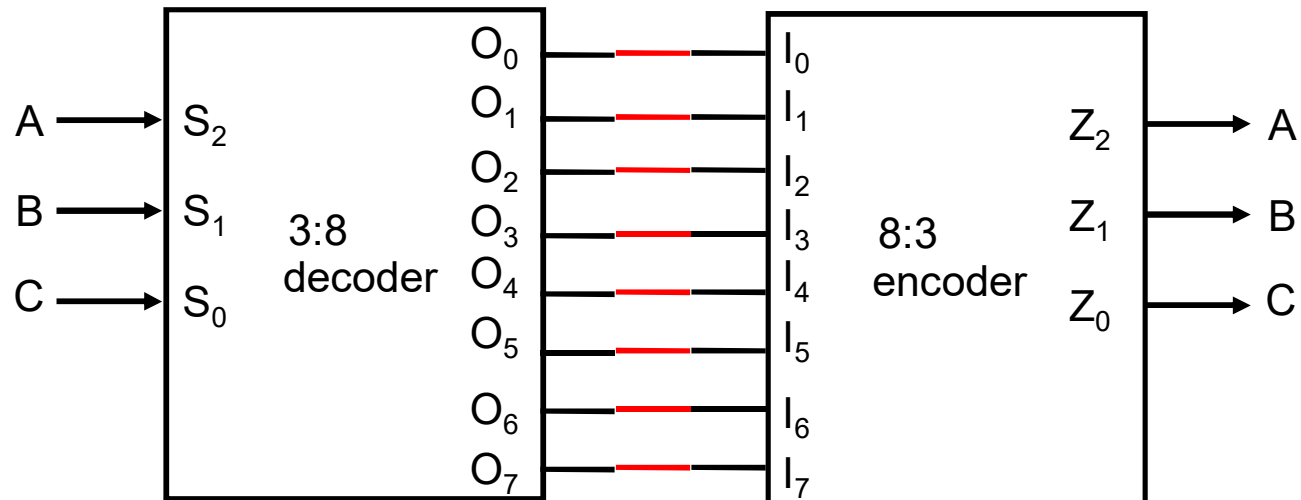
# Encoder

- **Encoder:**

- The reverse operation of a decoder
- Has  $2^n$  input lines and  $n$  output lines
- The output lines generate the binary equivalent of the input line whose value is 1



# Encoder



# Encoder Circuit Design

- **Example:**

- 8-3 Binary Encoder

Inputs								Outputs		
$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	$A_2$	$A_1$	$A_0$
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

$$A_0 = D_1 + D_3 + D_5 + D_7$$

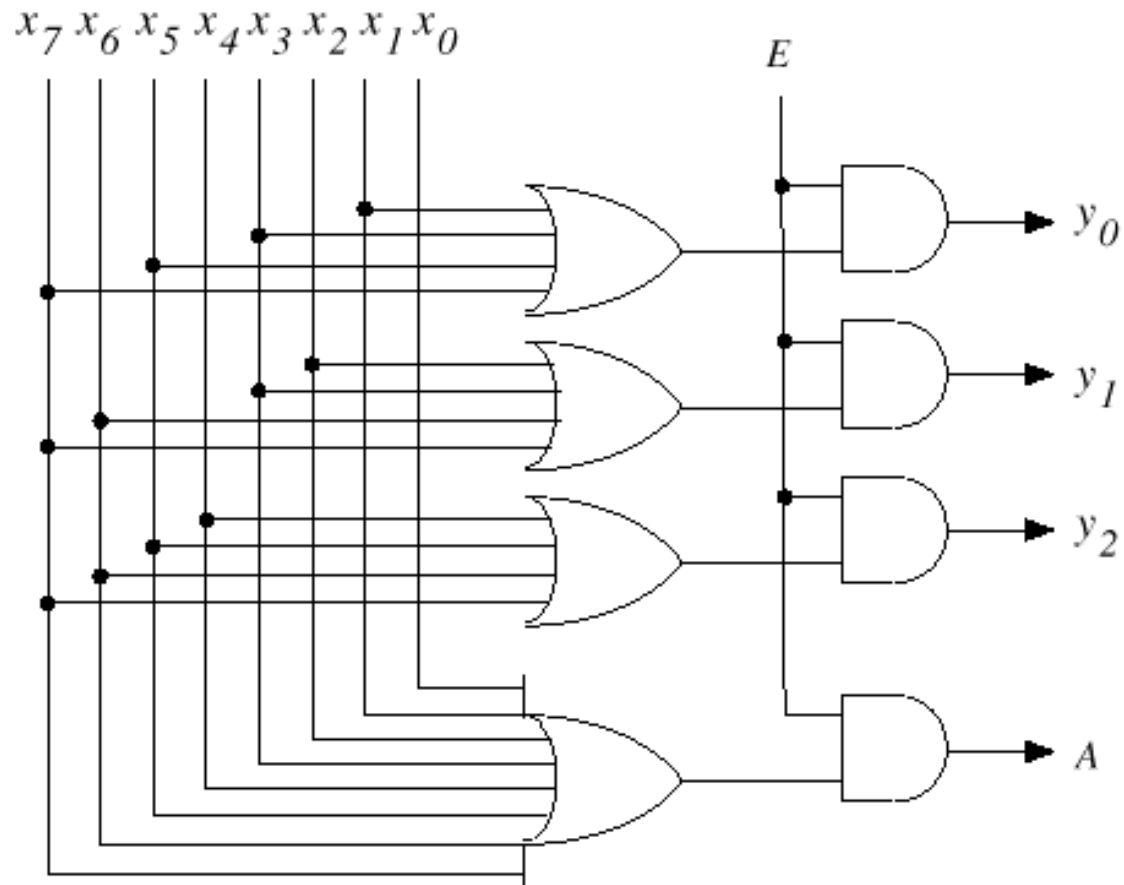
$$A_1 = D_2 + D_3 + D_6 + D_7$$

$$A_2 = D_4 + D_5 + D_6 + D_7$$



# Encoder Circuit

With Enable



With Acknowledge

# Encoder Design Issues

- The previous design has a major problem:
  - Only one input can be active at any given time
  - If two inputs are active simultaneously, the output produces an undefined combination
    - (for example, if  $D_3$  and  $D_6$  are 1 simultaneously, the output of the encoder will be 111.

$$A_0 = D_1 + D_3 + D_5 + D_7$$

$$A_1 = D_2 + D_3 + D_6 + D_7$$

$$A_2 = D_4 + D_5 + D_6 + D_7$$

May still be fine in particular applications! Example: next slide

# Encoder Application Example

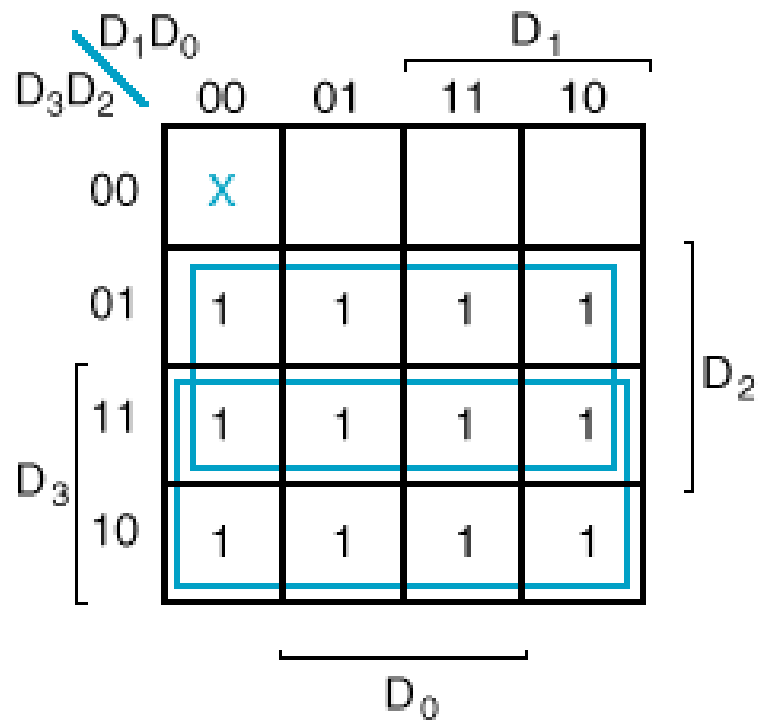
- Multiple contestant switch board
- Contestants have to push their buttons if they know the answer
- The winner's number will be shown
- One input from each contestant to the encoder
- Problems: too many inputs, small chance for simultaneous pushes

# Priority Encoder

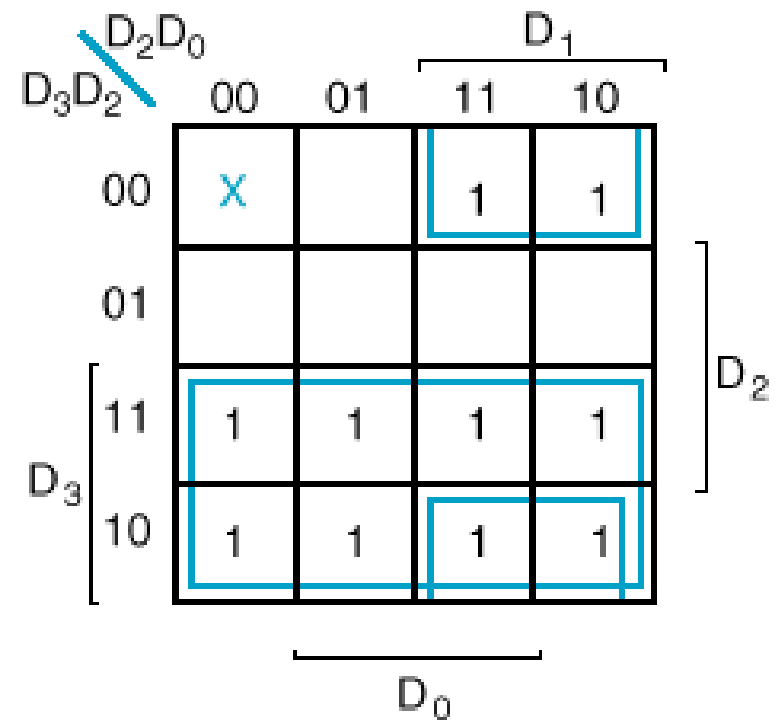
- Multiple asserted inputs are allowed; one has priority over all others.

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

# K-Maps

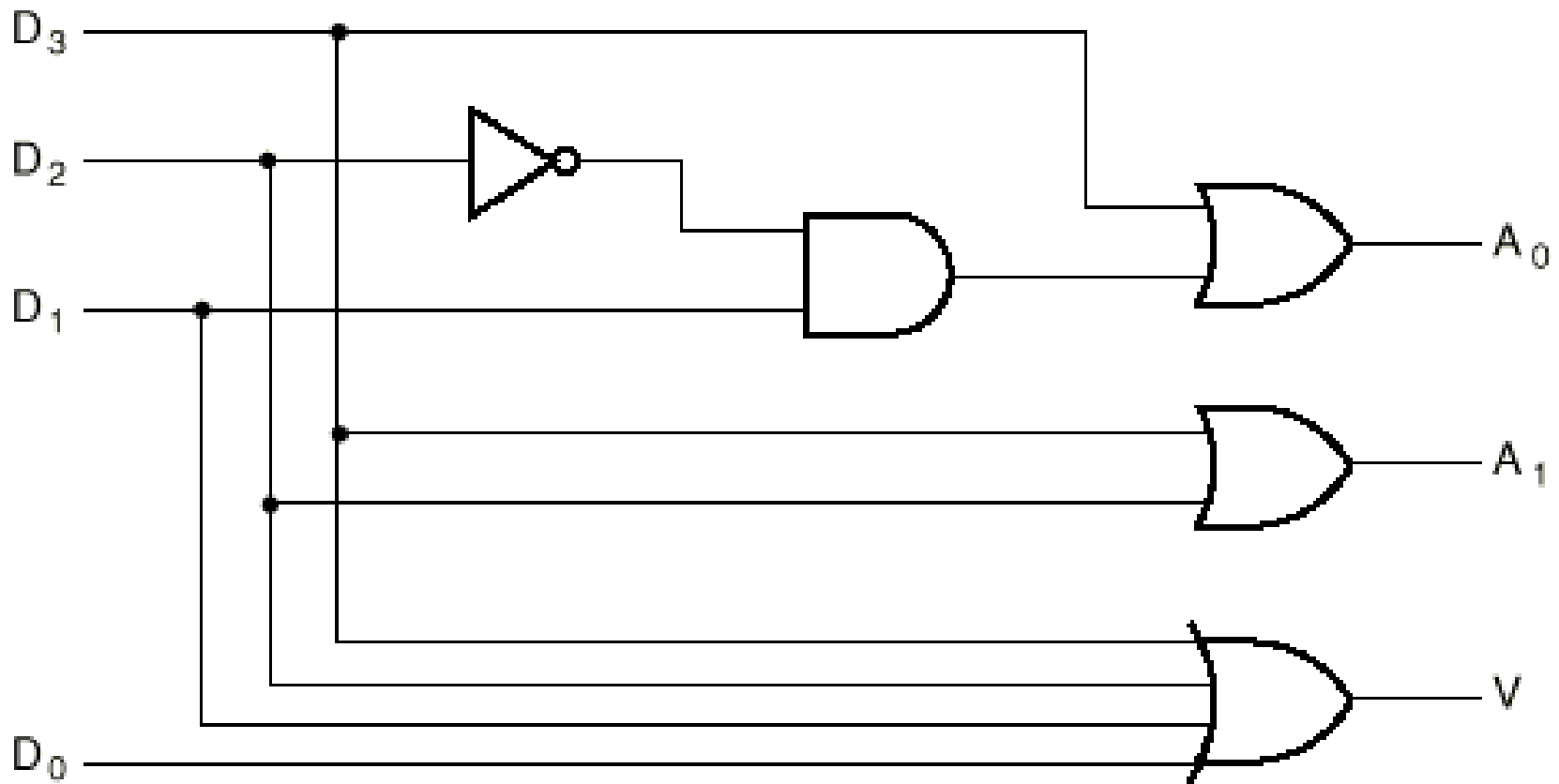


$$A_1 = D_2 + D_3$$



$$A_0 = D_3 + D_1\bar{D}_2$$

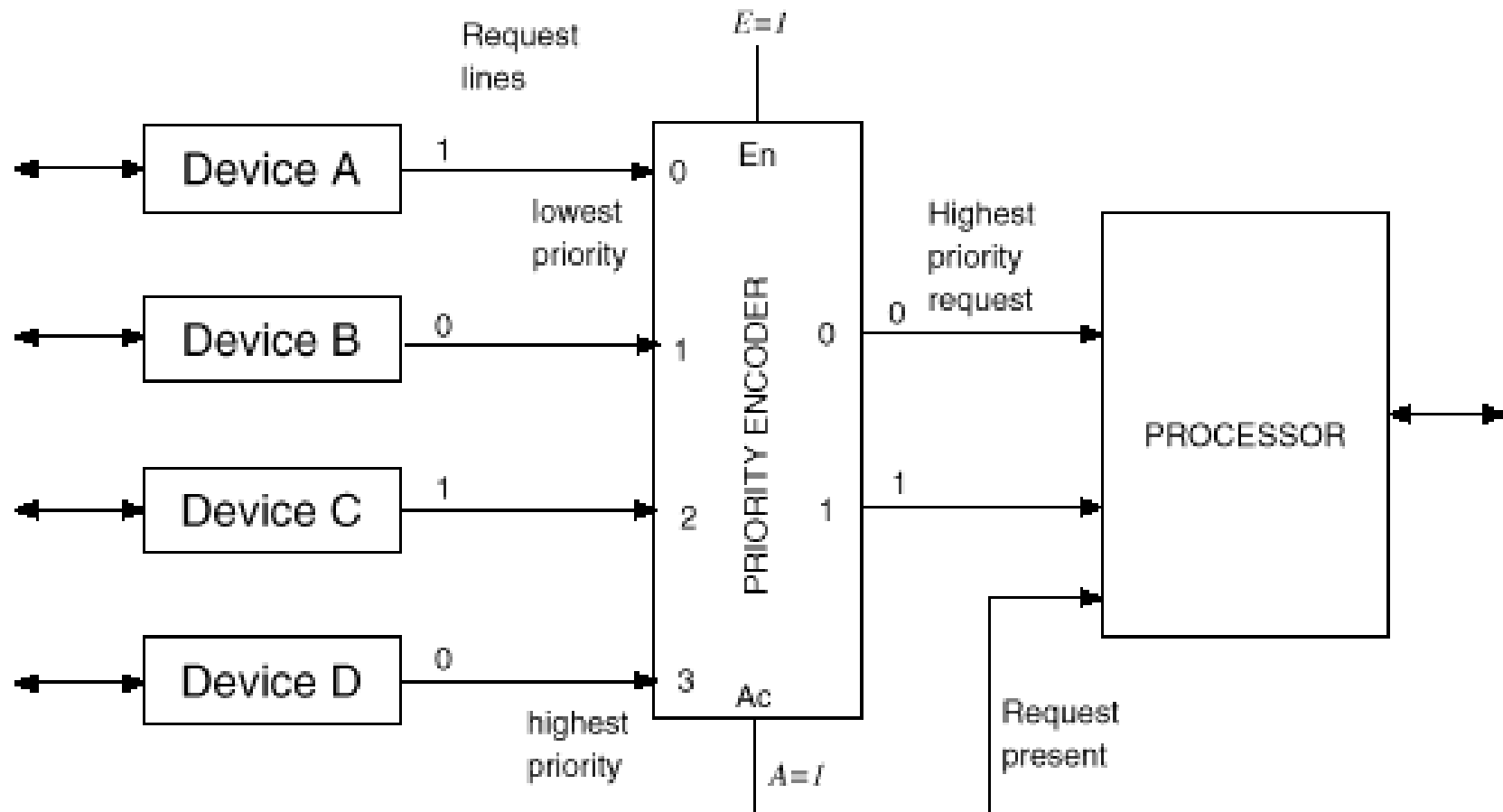
# Circuit



## 8-3 Priority Encoder

$A_0$	$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$	$A_7$	$Z_0$	$Z_1$	$Z_2$	$NR$
0	0	0	0	0	0	0	0	X	X	X	1
X	X	X	X	X	X	X	1	1	1	1	0
X	X	X	X	X	X	1	0	1	1	0	0
X	X	X	X	X	1	0	0	1	0	1	0
X	X	X	X	1	0	0	0	1	0	0	0
X	X	X	1	0	0	0	0	0	1	1	0
X	X	1	0	0	0	0	0	0	1	0	0
X	1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	0	0	0

# Priority Encoder: Application

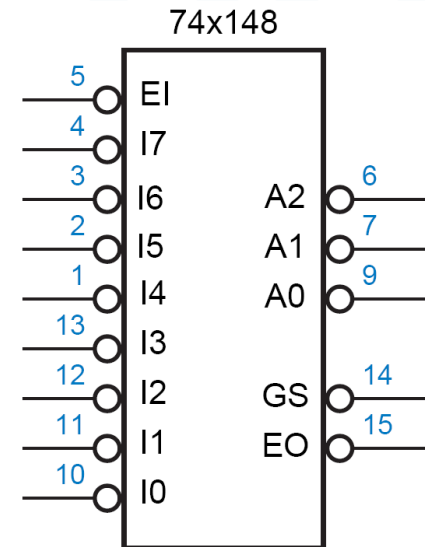




# 74x148

- **Features:**

- Inputs and outputs are active low
- EI must be asserted for any of its outputs to be asserted
- GS is asserted when the device is enabled and one or more of the request inputs is asserted (“Group Select” or “Got Something”)
- EO is an enable output designed to be connected to the EI input of another '148 that handles lower-priority requests
  - It is asserted if EI is asserted but no request input is asserted; thus, a lower-priority '148 may be enabled



# 74x148 Truth Table

<i>Inputs</i>									<i>Outputs</i>				
/EI	/I0	/I1	/I2	/I3	/I4	/I5	/I6	/I7	/A2	/A1	/A0	/GS	/EO
1	x	x	x	x	x	x	x	x	1	1	1	1	1
0	x	x	x	x	x	x	x	0	0	0	0	0	1
0	x	x	x	x	x	x	0	1	0	0	1	0	1
0	x	x	x	x	x	0	1	1	0	1	0	0	1
0	x	x	x	x	0	1	1	1	0	1	1	0	1
0	x	x	x	0	1	1	1	1	1	0	0	0	1
0	x	x	0	1	1	1	1	1	1	0	1	0	1
0	x	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0