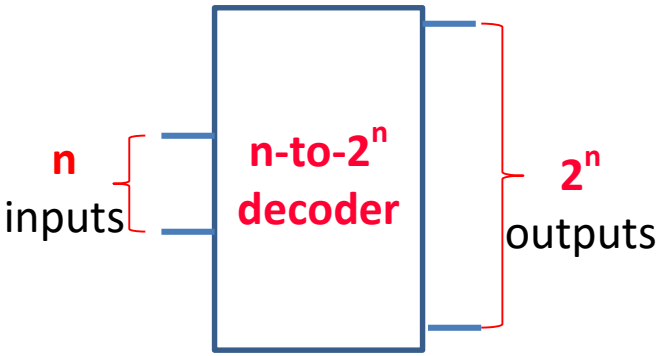
Digital Logic Design

Decoder & Encoder

Decoder

A **n-to-2^n decoder** takes an **n-bit** input and produces **2^n** outputs. The n inputs represent a binary number that determines which of the 2^n outputs is *uniquely* true.



Example:

- Reception counter: When you reach an Academic Institute

- Receptionist asks: Which Dept. to go?

- Based on your Specific answer, Receptionist redirects you to the specific building.

The job of the Decoder is to **Decode!**

-It knows what to do for a fixed question.

Use:

- Memory addressing

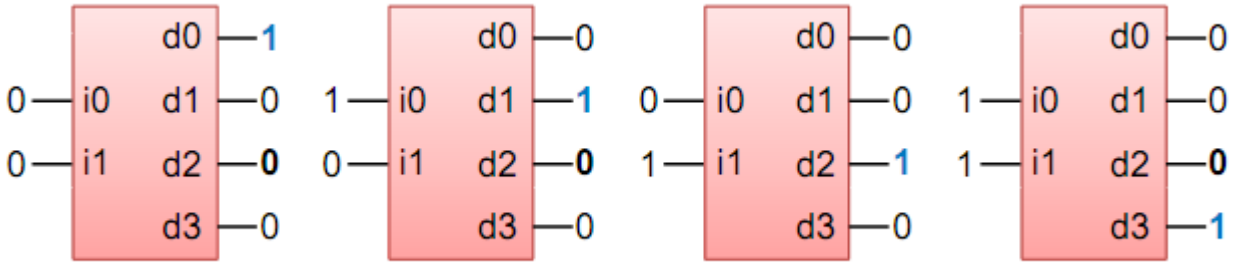
- Address to a particular location.

2-to-4 decoder

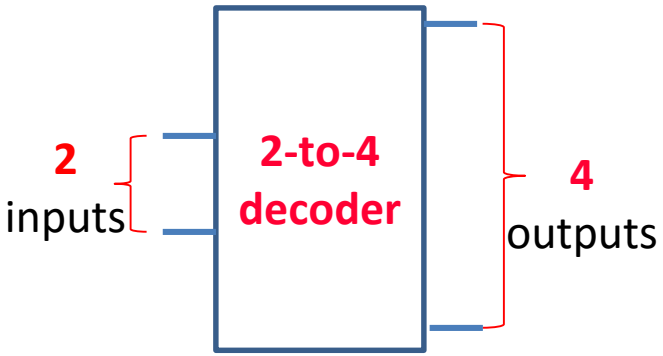
- A 2-to-4 decoder operates according to the following truth table.
  - The 2-bit input is called S1 S0, and the four outputs are Q0 Q1 Q2 Q3.
  - If the input is the binary number i, then output Qi is uniquely true.

S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

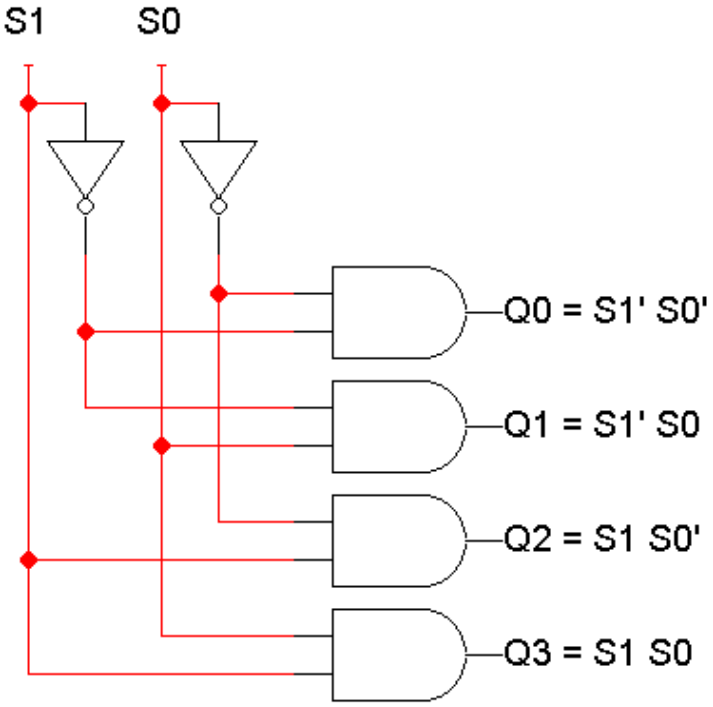
For instance, if the input **S1 S0 = 10** (decimal 2), then output **Q2 is true**, and **Q0, Q1, Q3 are all false**.
This circuit "decodes" a binary number into a "one-of-four" code.



Logic diagram of a 2-to-4 decoder



S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



- Q0 = S1' S0'
- Q1 = S1' S0
- Q2 = S1 S0'
- Q3 = S1 S0

- Decoding ONLY a specific sequence:

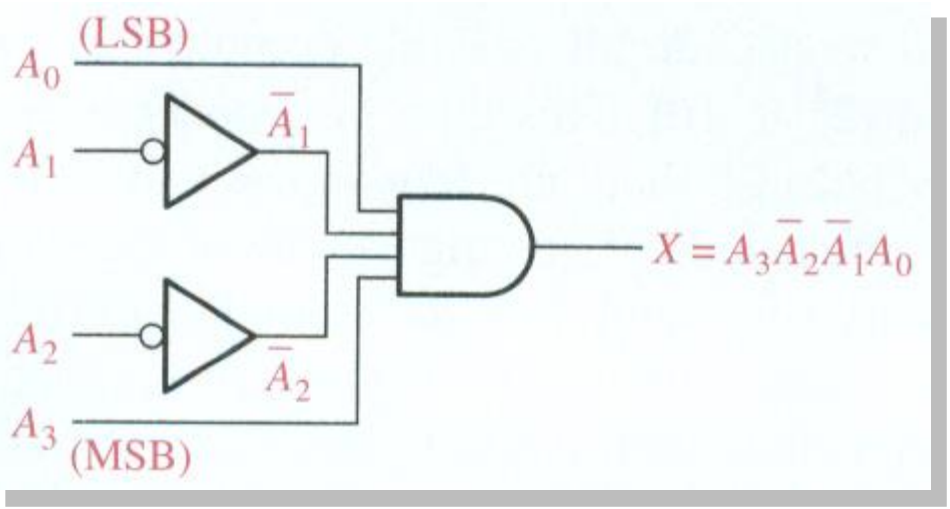
The output is 1 only when:

$A_0 = 1$

$A_1 = 0$

$A_2 = 0$

$A_3 = 1$



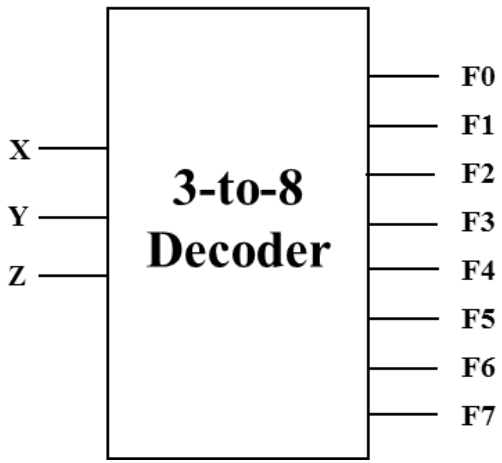
Use:

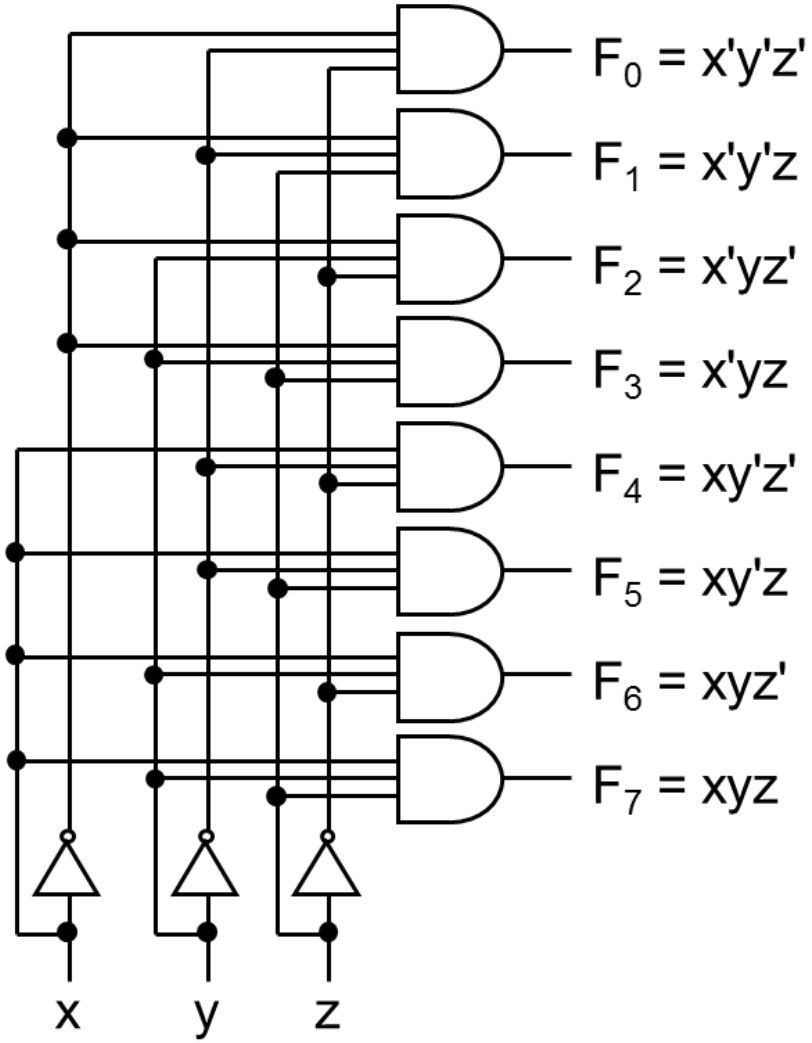
1) Encryption system,
2) Counter decoding…etc.

3-to-8 Binary Decoder

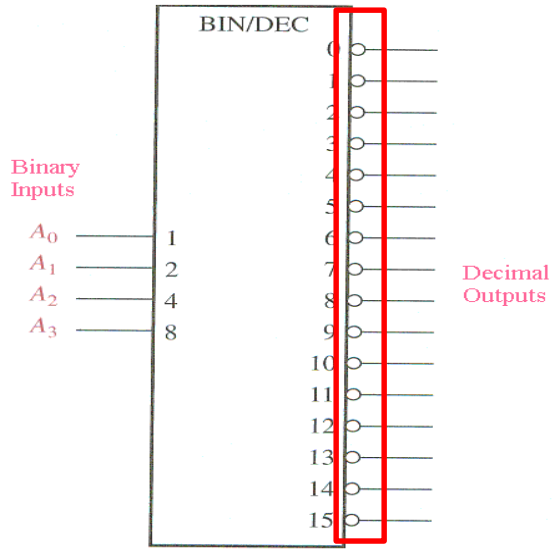
Truth Table:

x	y	z	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
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





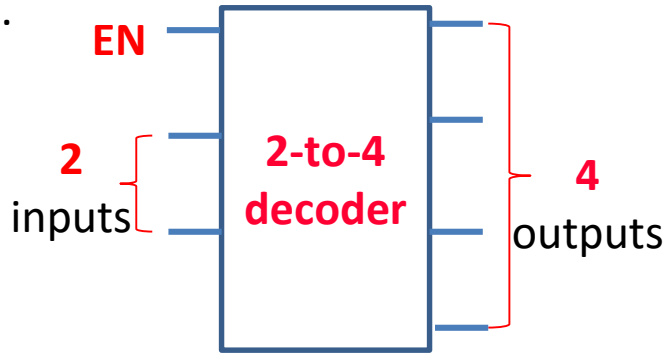
4-bit decoder



BINARY INPUTS				DECODING	OUTPUTS														
A_3	A_2	A_1	A_0	FUNCTION	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	$\bar{A}_3\bar{A}_2\bar{A}_1\bar{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	$\bar{A}_3\bar{A}_2\bar{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	0	$\bar{A}_3\bar{A}_2A_1\bar{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	$\bar{A}_3\bar{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
0	1	0	0	$\bar{A}_3A_2\bar{A}_1\bar{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
0	1	0	1	$\bar{A}_3A_2\bar{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
0	1	1	0	$\bar{A}_3A_2A_1\bar{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
0	1	1	1	$\bar{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
1	0	0	0	$A_3\bar{A}_2\bar{A}_1\bar{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
1	0	0	1	$A_3\bar{A}_2\bar{A}_1A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	0	1	0	$A_3\bar{A}_2A_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
1	0	1	1	$A_3\bar{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
1	1	0	0	$A_3A_2\bar{A}_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
1	1	0	1	$A_3A_2\bar{A}_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
1	1	1	0	$A_3A_2A_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Enable inputs

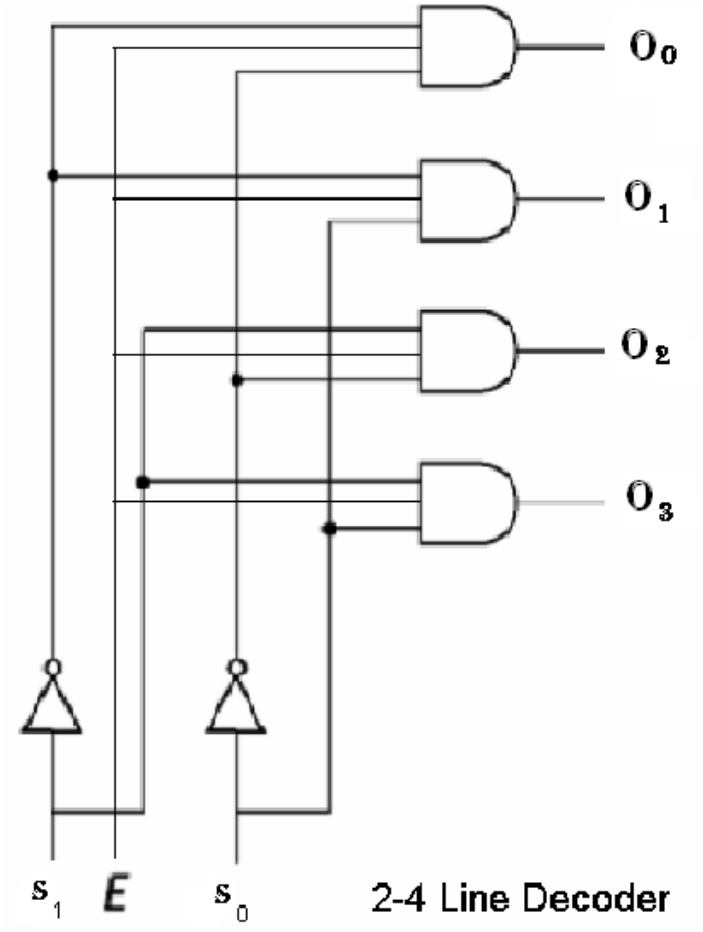
- Many devices have an additional **enable input**, which is used to "**activate**" or "**deactivate**" the device.



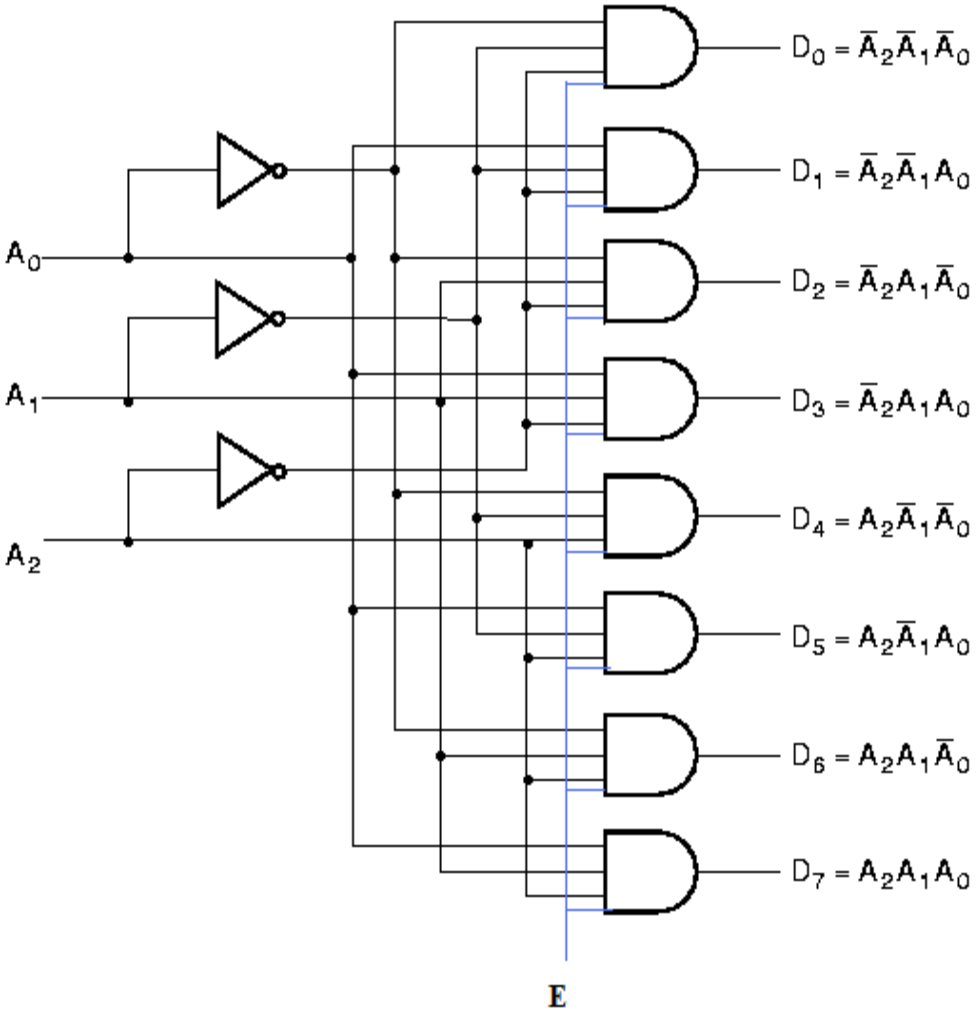
- For a decoder,
  - EN=0 "deactivates" the decoder. By convention, that means *all* of the decoder's outputs are 0.
  - EN=1 activates the decoder, so it behaves as specified earlier. Exactly one of the outputs will be 1.

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

2-to-4

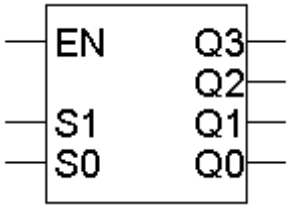

O_0
O_1
O_2
O_3
S_1
E
S_0
2-4 Line Decoder

3-to-8


A_0
A_1
A_2
$D_0 = \bar{A}_2\bar{A}_1\bar{A}_0$
$D_1 = \bar{A}_2\bar{A}_1A_0$
$D_2 = \bar{A}_2A_1\bar{A}_0$
$D_3 = \bar{A}_2A_1A_0$
$D_4 = A_2\bar{A}_1\bar{A}_0$
$D_5 = A_2\bar{A}_1A_0$
$D_6 = A_2A_1\bar{A}_0$
$D_7 = A_2A_1A_0$
E

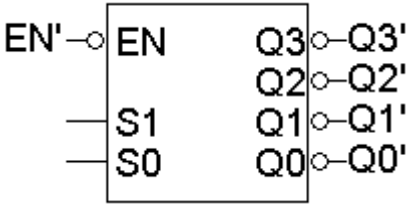
A variation of the standard decoder

- The decoders we've seen so far are **active-high** decoders.



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

- An **active-low** decoder is the same thing, but 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

So what good is a decoder?

- Do the truth table and equations look familiar?

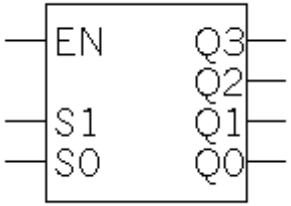
S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Q0 = S1′ S0′
Q1 = S1′ S0
Q2 = S1 S0′
Q3 = S1 S0

- Decoders are sometimes called **minterm generators**.
  - For each of the input combinations, exactly one output is true.
  - Each output equation contains all of the input variables.
  - These properties hold for all sizes of decoders.
- This means that you can implement arbitrary functions with decoders. If you have a sum of minterms equation for a function, you can easily use a decoder (a minterm generator) to implement that function.

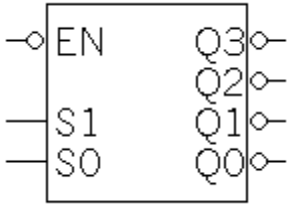
Mysteriously Similar

- **Active-high** decoders generate ***minterms***, as we've already seen.



Q3 = S1 S0
Q2 = S1 S0'
Q1 = S1' S0
Q0 = S1' S0'

- The output equations for an **active-low** decoder are mysteriously similar, yet somehow different.



Q3' = (S1 S0)' = S1' + S0'
Q2' = (S1 S0')' = S1' + S0
Q1' = (S1' S0)' = S1 + S0'
Q0' = (S1' S0')' = S1 + S0

- It turns out that **active-low** decoders generate ***maxterms*.**

Implementing Functions using Decoders

Design example: addition

- Let's make a circuit that adds three 1-bit inputs X, Y and Z.
- We will need two bits to represent the total; let's call them C and S, for "carry" and "sum." Note that C and S are two separate functions of the same inputs X, Y and Z.

0 + 1 + 1 = 10 →

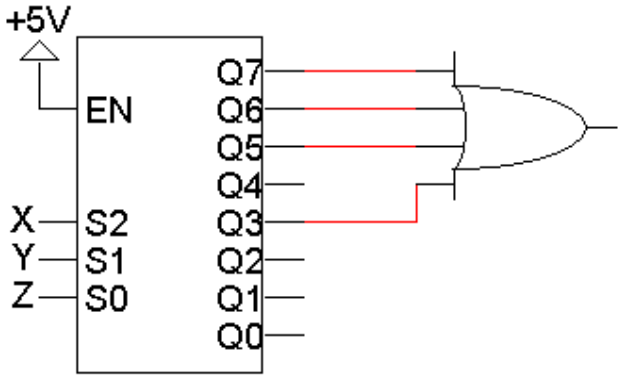
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$C(X,Y,Z) = \Sigma\ (3,5,6,7)$
$S(X,Y,Z) = \Sigma\ (1,2,4,7)$

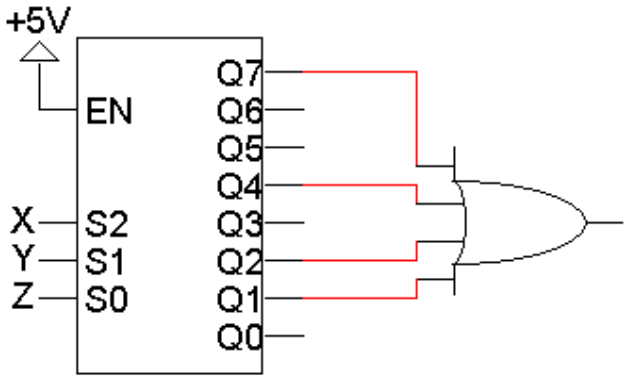
← 1 + 1 + 1 = 11

Decoder-based adder

- Here, two 3-to-8 decoders implement C and S as sums of minterms.



$C(X,Y,Z) = \Sigma(3,5,6,7)$

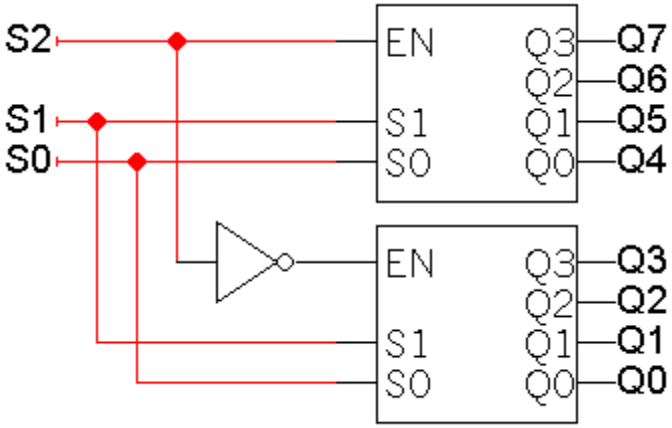


$S(X,Y,Z) = \Sigma(1,2,4,7)$

- The "+5V" symbol ("5 volts") is how you represent a constant 1 or true in Logic Works. It has been used here so that the decoders are always active.

Decoder expansion

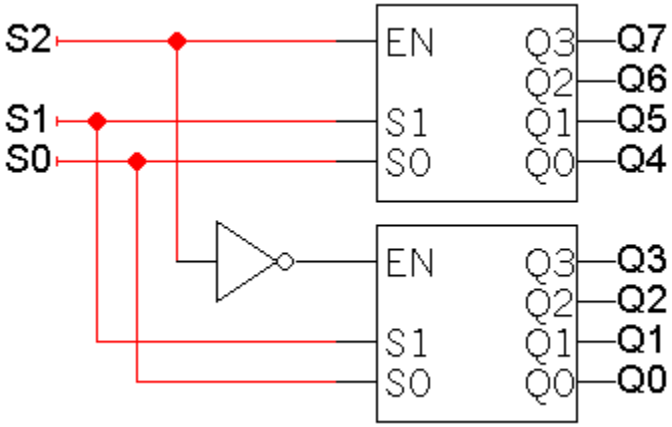
- Combine two or more small decoders with **enable** inputs to form a larger decoder.
- Here a 3-to-8 decoder has been constructed from **two** 2-to-4 decoders:



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

Modularity

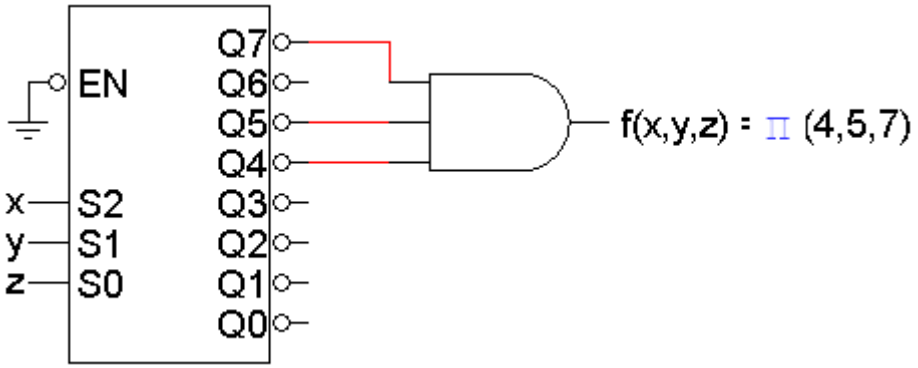
- Be careful not to confuse the “inner” inputs and outputs of the 2-to-4 decoders with the “outer” inputs and outputs of the 3-to-8 decoder (which are in boldface).
- This is similar to having several functions in a program which all use a formal parameter “x”.



- You could verify that this circuit is a 3-to-8 decoder, by using equations for the 2-to-4 decoders to derive equations for the 3-to-8.

Active-low decoder example

- So we can use active-low decoders to implement arbitrary functions too, but as a product of maxterms.
- For example, here is an implementation of the function, $f(x,y,z) = \Pi(4,5,7)$, using an active-low decoder.



- The "ground" symbol connected to EN represents logical 0, so this decoder is always enabled.
- Remember that you need an AND gate for a product of sums.

Use two 3 to 8 decoders to make 4 to 16 decoder

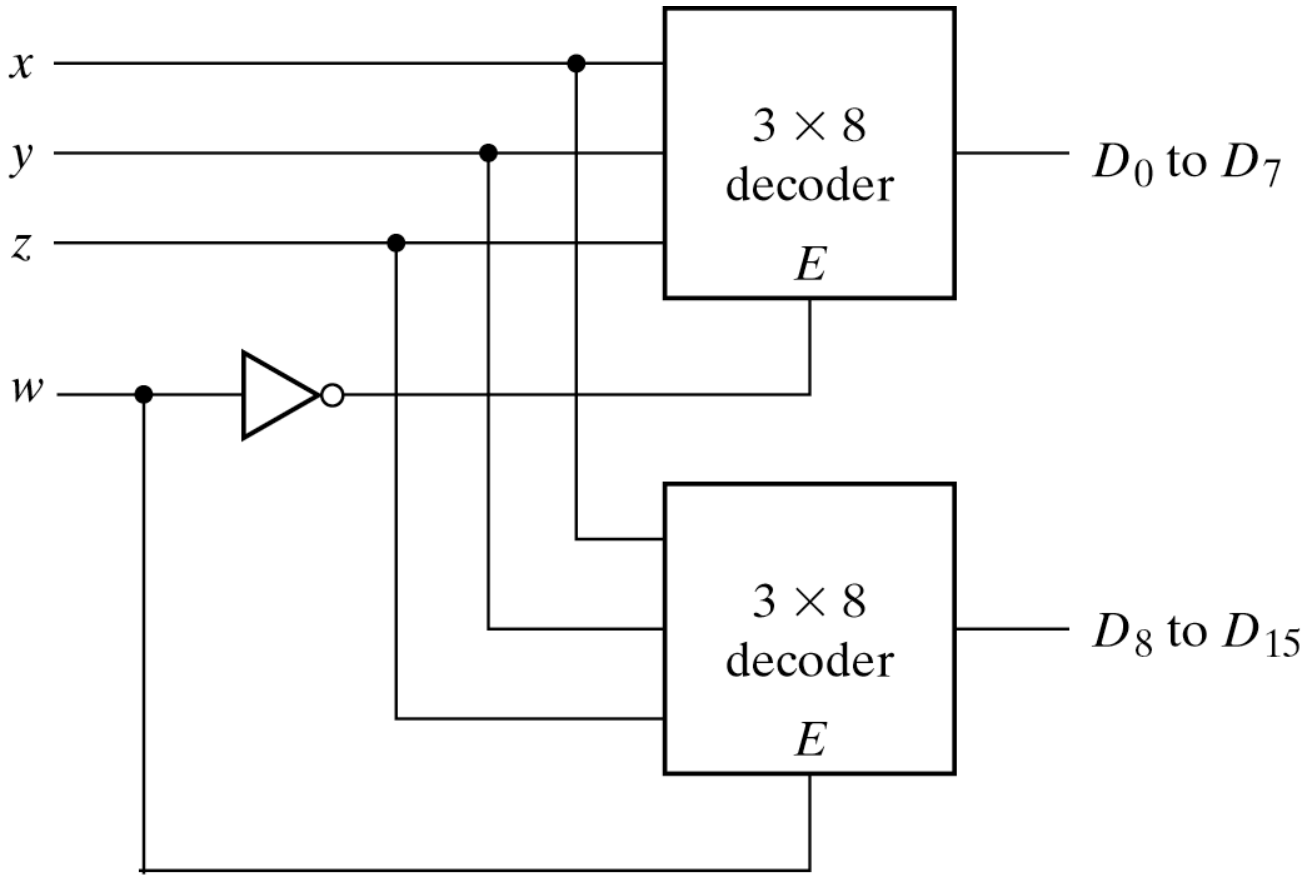
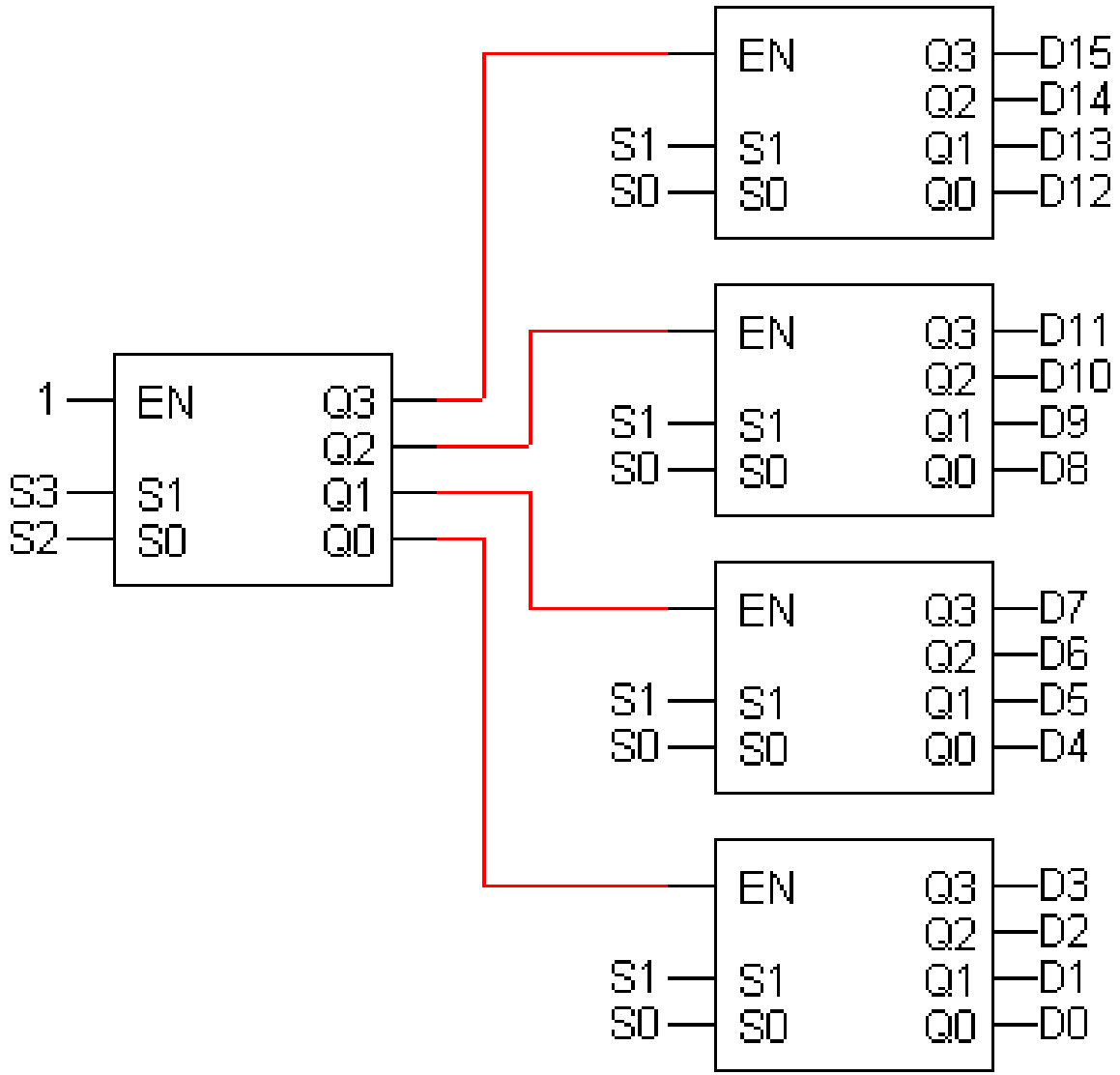


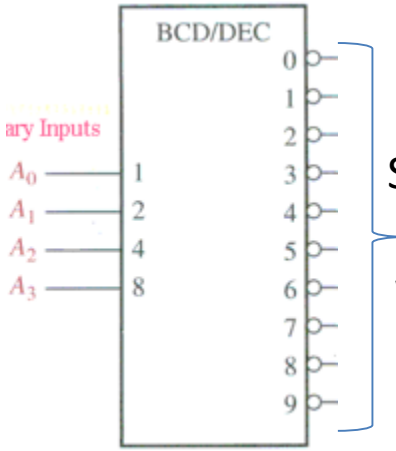
Fig. 4-20 4 × 16 Decoder Constructed with Two 3 × 8 Decoders

4-to-16 decoder using only 2-to-4 decoders (no gates)



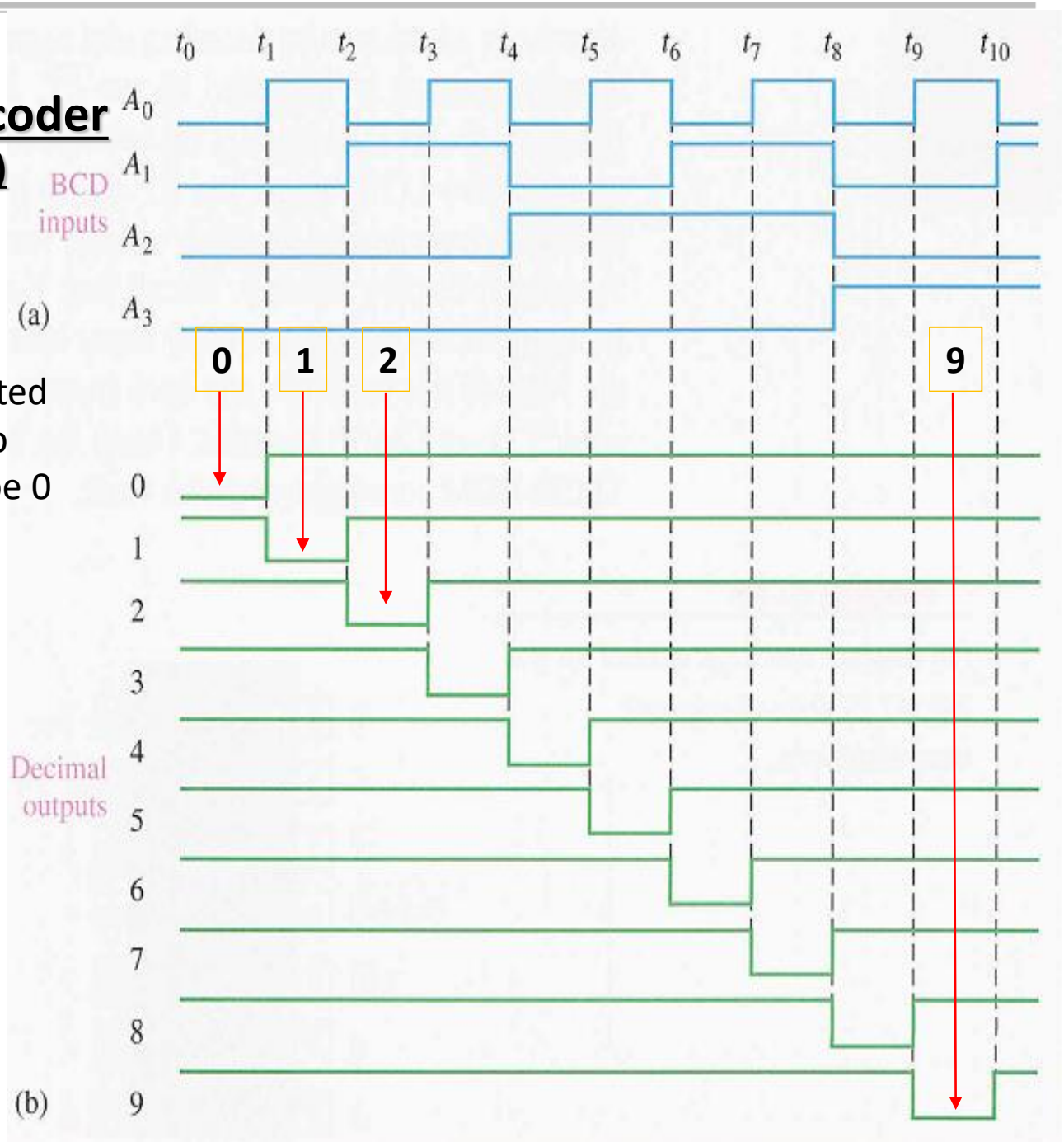
BCD-to-decimal decoder

(Active Low Output)



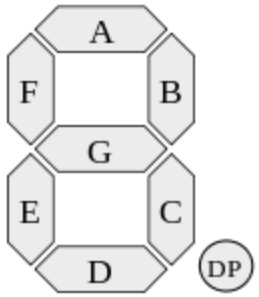
Selected o/p will be 0

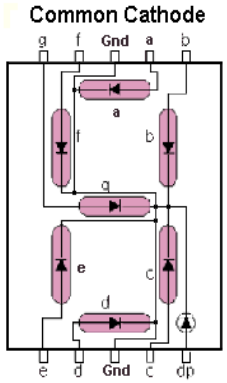
DECIMAL DIGIT	BCD CODE A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

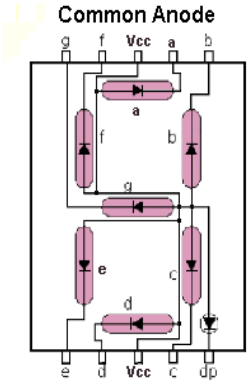


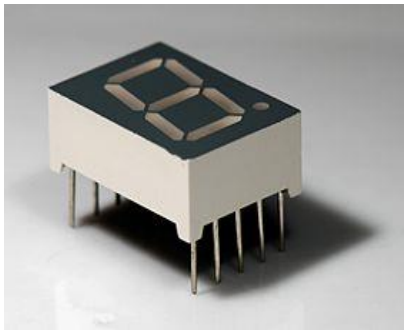
BCD-to-7-segement decoder

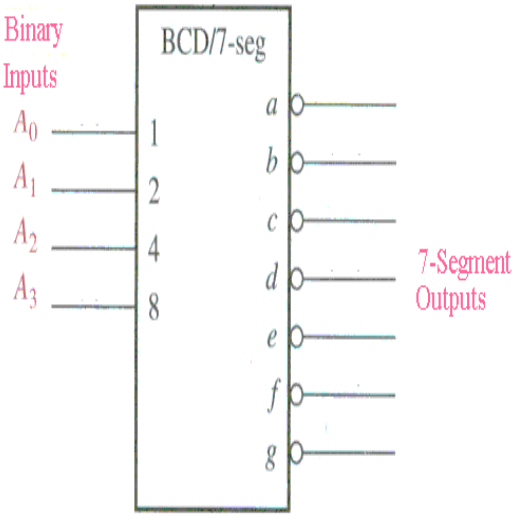
DECIMAL DIGIT	INPUTS				SEGMENT OUTPUTS						
	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	X	X	X	X	X	X	X
11	1	0	1	1	X	X	X	X	X	X	X
12	1	1	0	0	X	X	X	X	X	X	X
13	1	1	0	1	X	X	X	X	X	X	X
14	1	1	1	0	X	X	X	X	X	X	X
15	1	1	1	1	X	X	X	X	X	X	X

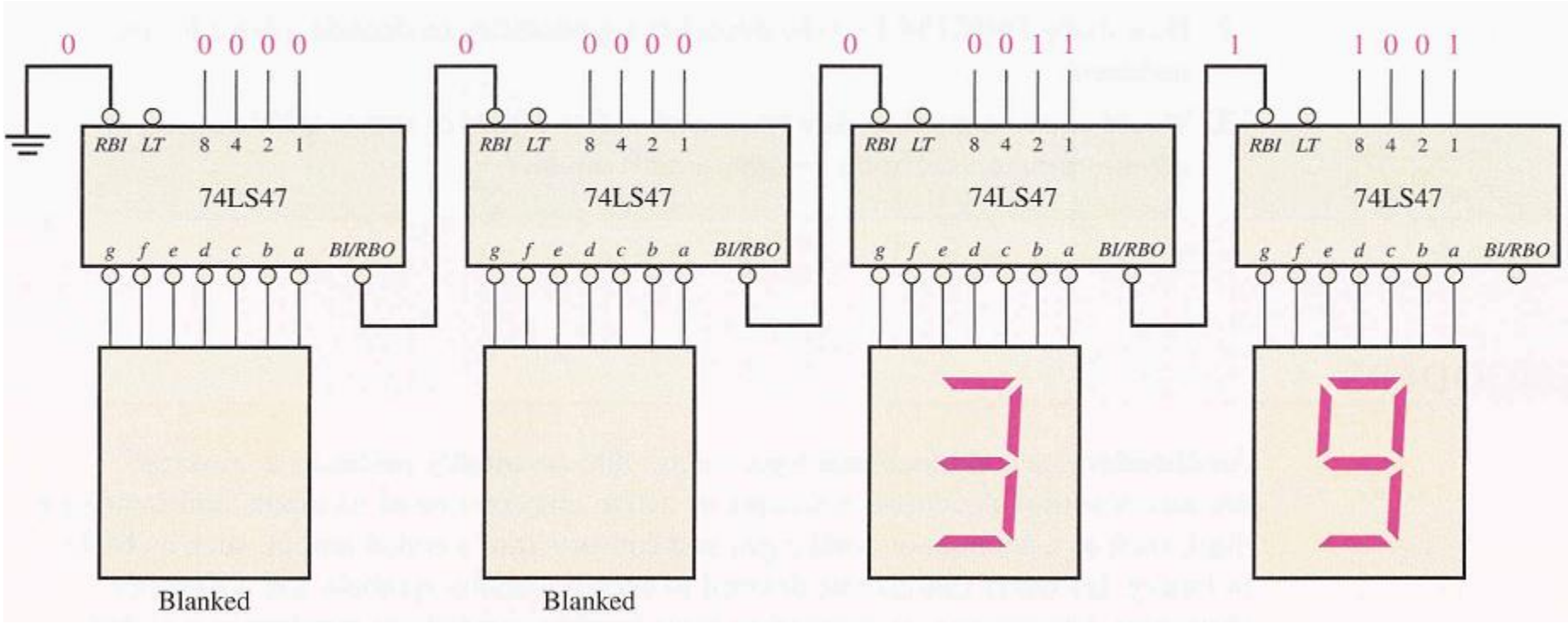











0
0 0 0 0
RBI LT 8 4 2 1
74LS47
g f e d c b a BI/RBO
Blanked
0
0 0 0 0
RBI LT 8 4 2 1
74LS47
g f e d c b a BI/RBO
Blanked
0
0 0 1 1
RBI LT 8 4 2 1
74LS47
g f e d c b a BI/RBO
1
1 0 0 1
RBI LT 8 4 2 1
74LS47
g f e d c b a BI/RBO

Summary

- A n-to-2^n decoder generates the minterms of a n-variable function.
  - As such, decoders can be used to implement arbitrary functions.
- Some variations of the basic decoder include:
  - Adding an enable input.
  - Using active-low inputs and outputs to generate maxterms.
- We also talked about:
  - Applying our circuit analysis and design techniques to understand and work with decoders.
  - Using block symbols to encapsulate common circuits like decoders.
  - Building larger decoders from smaller ones.

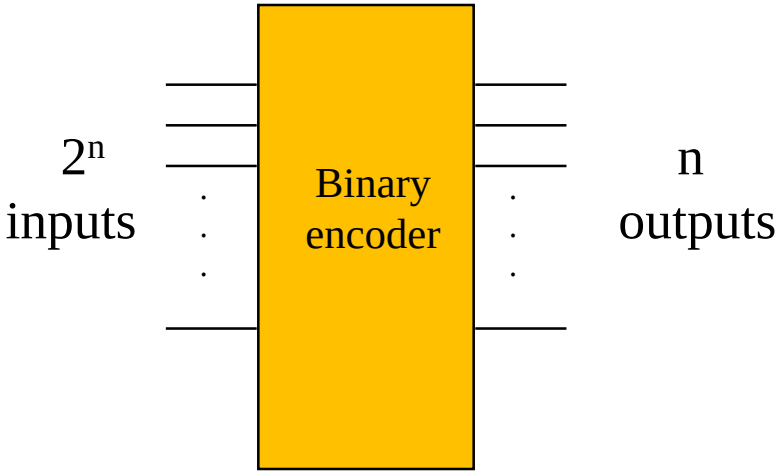
Encoders

An **Encoder** is a combinational logic circuit that performs a "**reverse**" decoder function.

An Encoder accepts an active level on one of ots inputs representing a digit, such as a decimal or octal digit, and converts it to a coded output, such as BCD or binary.

Encoders can also be devised to encode various symbols and alphabetic characters. The process of converting from familiar symbols or numbers to a coded format is called **Encoding**.

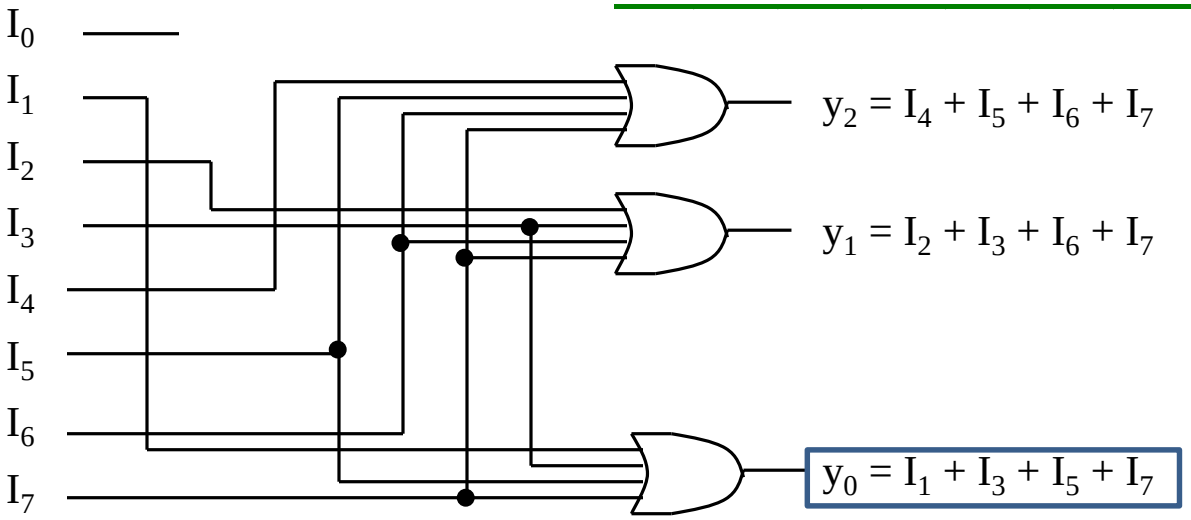
2^n-to-n Encoder:

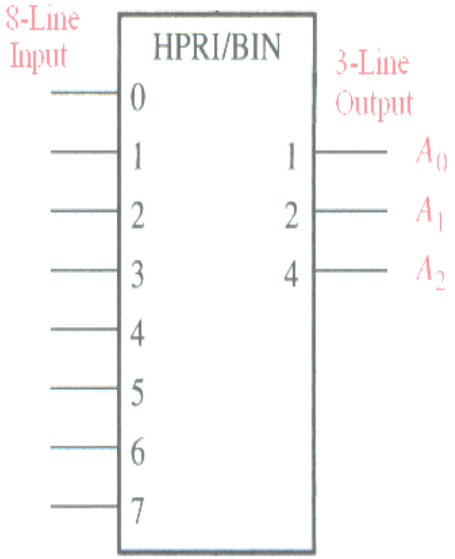


8-to-3 Binary Encoder

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

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





8-to-3 Priority Encoder

- What if more than one input line has a value of 1?

Example:

- For the above mentioned problem, let's give priority to higher bits
- Ignore "lower priority" inputs.
- The sequence is:

7>6>5>4>3>2>1>0

- Idle indicates that no input is a 1.

Inputs								Outputs			
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	y_2	y_1	y_0	Idle
0	0	0	0	0	0	0	0	x	x	x	1
1	0	0	0	0	0	0	0	0	0	0	0
X	1	0	0	0	0	0	0	0	0	1	0
X	X	1	0	0	0	0	0	0	1	0	0
X	X	X	1	0	0	0	0	0	1	1	0
X	X	X	X	1	0	0	0	1	0	0	0
X	X	X	X	X	1	0	0	1	0	1	0
X	X	X	X	X	X	1	0	1	1	0	0
X	X	X	X	X	X	X	1	1	1	1	0

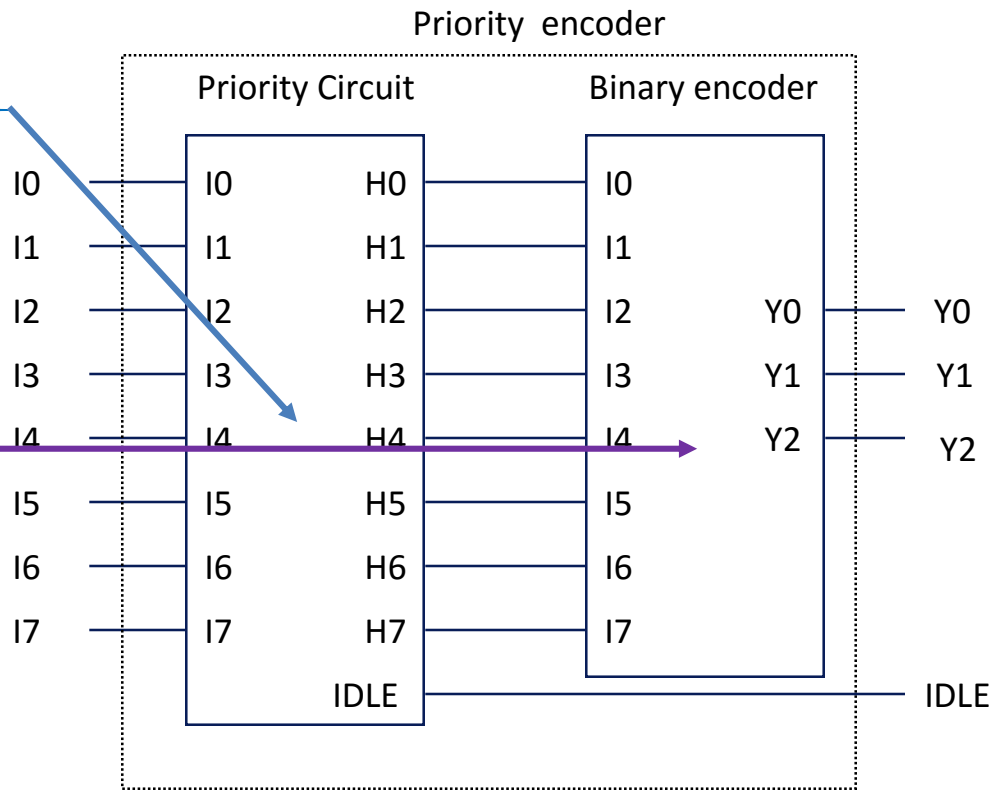
Priority Encoder (8 to 3 encoder)

- Assign priorities to the inputs
- When more than one inputs are asserted, the output generates the code of the input with the highest priority: 7>6>5>4>3>2>1>0
- Priority Encoder :

  H7=I7 (Highest Priority)
  H6=I6.I7'
  H5=I5.I6'.I7'
  H4=I4.I5'.I6'.I7'
  H3=I3.I4'.I5'.I6'.I7'
  H2=I2.I3'.I4'.I5'.I6'.I7'
  H1=I1. I2'.I3'.I4'.I5'.I6'.I7'
  H0=I0.I1'. I2'.I3'.I4'.I5'.I6'.I7'
  IDLE= I0'.I1'. I2'.I3'.I4'.I5'.I6'.I7'

- Encoder

  Y0 = I1 + I3 + I5 + I7
  Y1 = I2 + I3 + I6 + I7
  Y2 = I4 + I5 + I6 + I7



Priority Encoder (Irregular sequence)

- Assign priorities to the inputs in the following order:

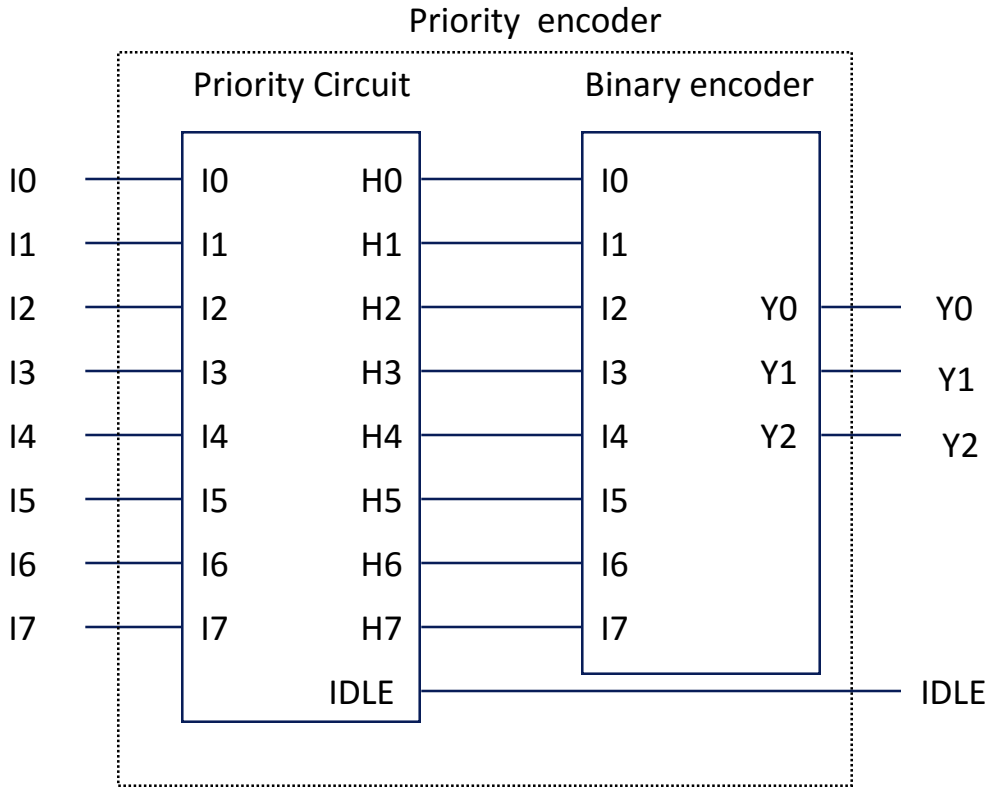
  2>5>3>4>6>7>1>0

- Priority Encoder :

  H2=I2 (Highest Priority)
  H5=I5.I2'
  H3=I3.I5'.I2'
  H4=I4. I3'.I5'.I2'
  H6=I6. I4'. I3'.I5'.I2'
  H7=I7. I6'. I4'. I3'.I5'.I2'
  H1=I1. I7'. I6'. I4'. I3'.I5'.I2'
  H0=I0. I1'. I7'. I6'. I4'. I3'.I5'.I2'
  IDLE= I0'.I1'. I2'.I3'.I4'.I5'.I6'.I7'

- Encoder

  Y0 = I1 + I3 + I5 + I7
  Y1 = I2 + I3 + I6 + I7
  Y2 = I4 + I5 + I6 + I7



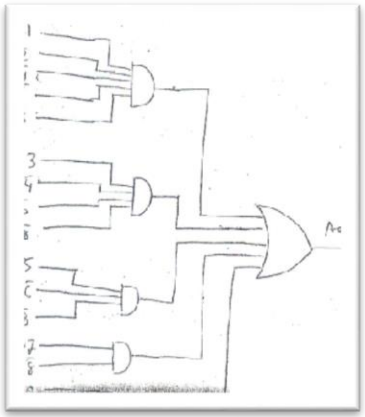
The Decimal - to - BCD Priority Encder:

Optional

Let Priority is given to the higher order digits. Requirements to activate A0:

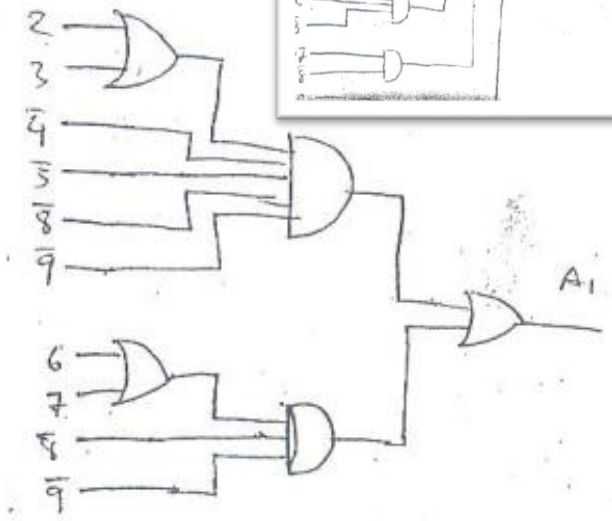
1) A_0 is HIGH if 1 is HIGH and 2,4,6,8 LOW
   A_0 is HIGH if 3 is HIGH and 4,6,8 LOW
   A_0 is HIGH if 5 is HIGH and 6,8 LOW
   A_0 is HIGH if 7 is HIGH and 8 LOW
   A_0 is HIGH if 9 is HIGH

Therefore, $A_0 = 1.2'.4'.6'.8' + 3.4'.6'.8' + 5.6'.8' + 7.8' + 9$



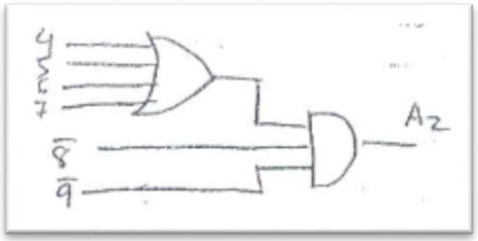
2) A_1 is HIGH if 2 is HIGH and 4,5,8,9 LOW
   A_1 is HIGH if 3 is HIGH and 4,5,8,9 LOW
   A_1 is HIGH if 6 is HIGH and 8,9 LOW
   A_1 is HIGH if 7 is HIGH and 8,9 LOW

Therefore, $A_1 = (2+3)4'.5'.8'.\ 9' + (6+7)8'.\ 9'$



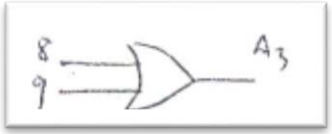
3) A_2 is HIGH if 4 is HIGH and 8,9 LOW
   A_2 is HIGH if 5 is HIGH and 8,9 LOW
   A_2 is HIGH if 6 is HIGH and 8,9 LOW
   A_2 is HIGH if 7 is HIGH and 8,9 LOW

Therefore, $A_2 = (4+5+6+7)8'.\ 9'$



4) A_3 is HIGH if 8 &9 are HIGH

Therefore, $A_3 = 8+9$



Reference:

Mixed contents from books by Floyd; Mano; Vahid And Howard.

Acknowledgement:

Nafiz Ahmed Chisty


AMERICAN INTERNATIONAL UNIVERSITY
PRAESIDIUM
DISCIPLINA
1994
BANGLADESH

Thanks