CSE 201: Digital Logic Design

BCD Adder, Magnitude Comparator, Decoder, Demultiplexer

Prepared By

Lec Sumaiya Afroz Mila

CSE, MIST

BINARY CODED DECIMAL

- Binary coded decimal (BCD) is a system of writing numerals that assigns a four-digit binary code to each digit 0 through 9 in a decimal (base-10) numeral.

- Numbers larger than 9, having two or more digits in the decimal system, are expressed digit by digit. For example, the BCD rendition of the number 1895, (base-10) is

0001 1000 1001 0101

Decimal	BCD	Binary
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001

BCD ADDER

- We will perform addition operation of two decimal numbers in BCD
- Each input does not exit 9
- Sum output will be highest: 19

```
  9(augend)
+9(addend)
+1(input carry)
---------------
 19
```

- The binary value and BCD value of the augend and addend are same
- Apply the BCD value of the augend and the addend to a 4-bit binary adder
- Adder will form the sum in binary and produce a result that may range from 0 to 19
- This binary result should be converted to BCD

BCD ADDER

8

9

Binary Sum					BCD Sum				
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0
0	0	0	1	1	0	0	0	1	1
0	0	1	0	0	0	0	1	0	0
0	0	1	0	1	0	0	1	0	1
0	0	1	1	0	0	0	1	1	0
0	0	1	1	1	0	0	1	1	1
0	1	0	0	0	0	1	0	0	0
0	1	0	0	1	0	1	0	0	1
0	1	0	1	0	1	0	0	0	0
0	1	0	1	1	1	0	0	0	1
0	1	1	0	0	1	0	0	1	0
0	1	1	0	1	1	0	0	1	1
0	1	1	1	0	1	0	1	0	0
0	1	1	1	1	1	0	1	0	1
1	0	0	0	0	1	0	1	1	0
1	0	0	0	1	1	0	1	1	1
1	0	0	1	0	1	1	0	0	0
1	0	0	1	1	1	1	0	0	1

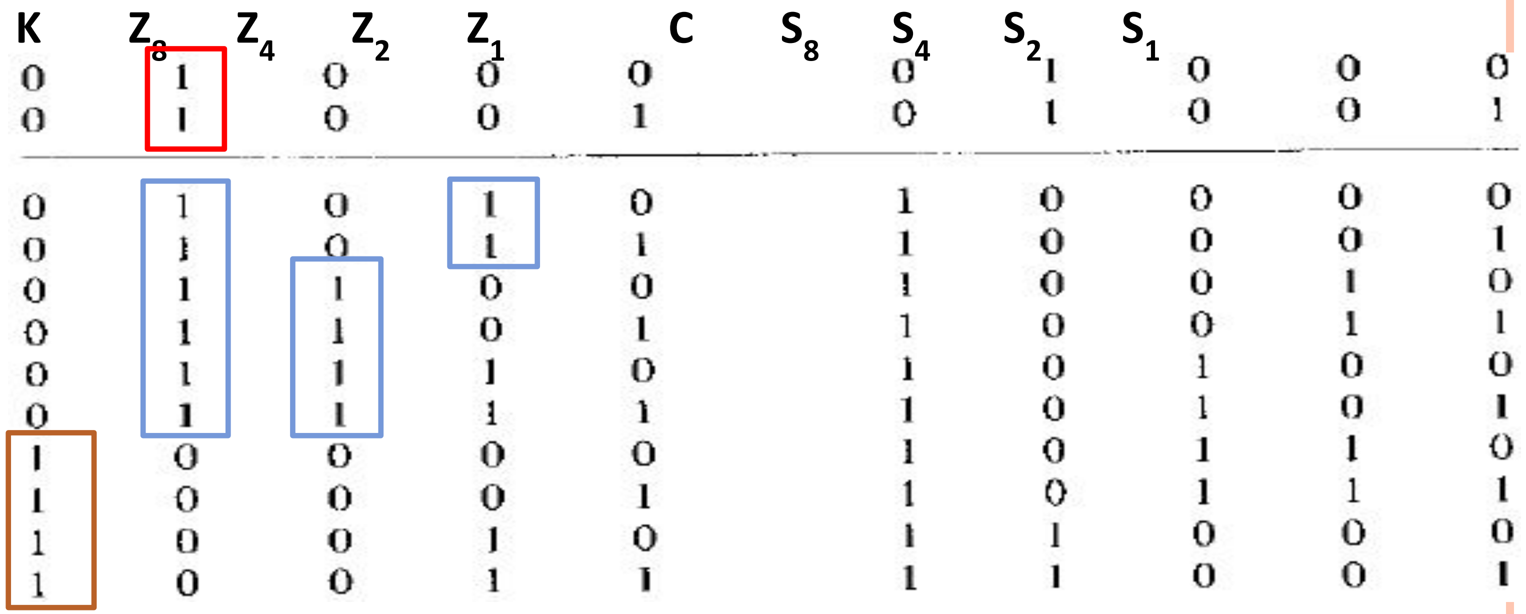
BCD ADDER

- From the table, when binary sum ≤ 1001(9_{10}), corresponding BCD is identical – **No conversion needed**
- When binary sum > 1001, it becomes non-valid as BCD – conversion needed
  - For binary sum>1001, we have to convert it to a valid BCD representation
  - When binary sum is 1010, decimal is 10 – correct BCD representation will be

    0001 0000
  - To convert 1010 to 10000, we have to add $(0110)_2$ or $(6)_{10}$ with 1010
  - Similarly if we add $(0110)_2$ or $(6)_{10}$ with 1011……10011, we will get the corresponding BCD representation

BCD ADDER

- Now we have to design the circuit.
- We have to design a circuit that will detect when conversion should be performed and when not.
- From the table, it is visible that, conversion is needed when the binary sum has an output carry K=1.
- Other 6 combinations From 1010 to 1111 that Need conversion have a 1 in position z8



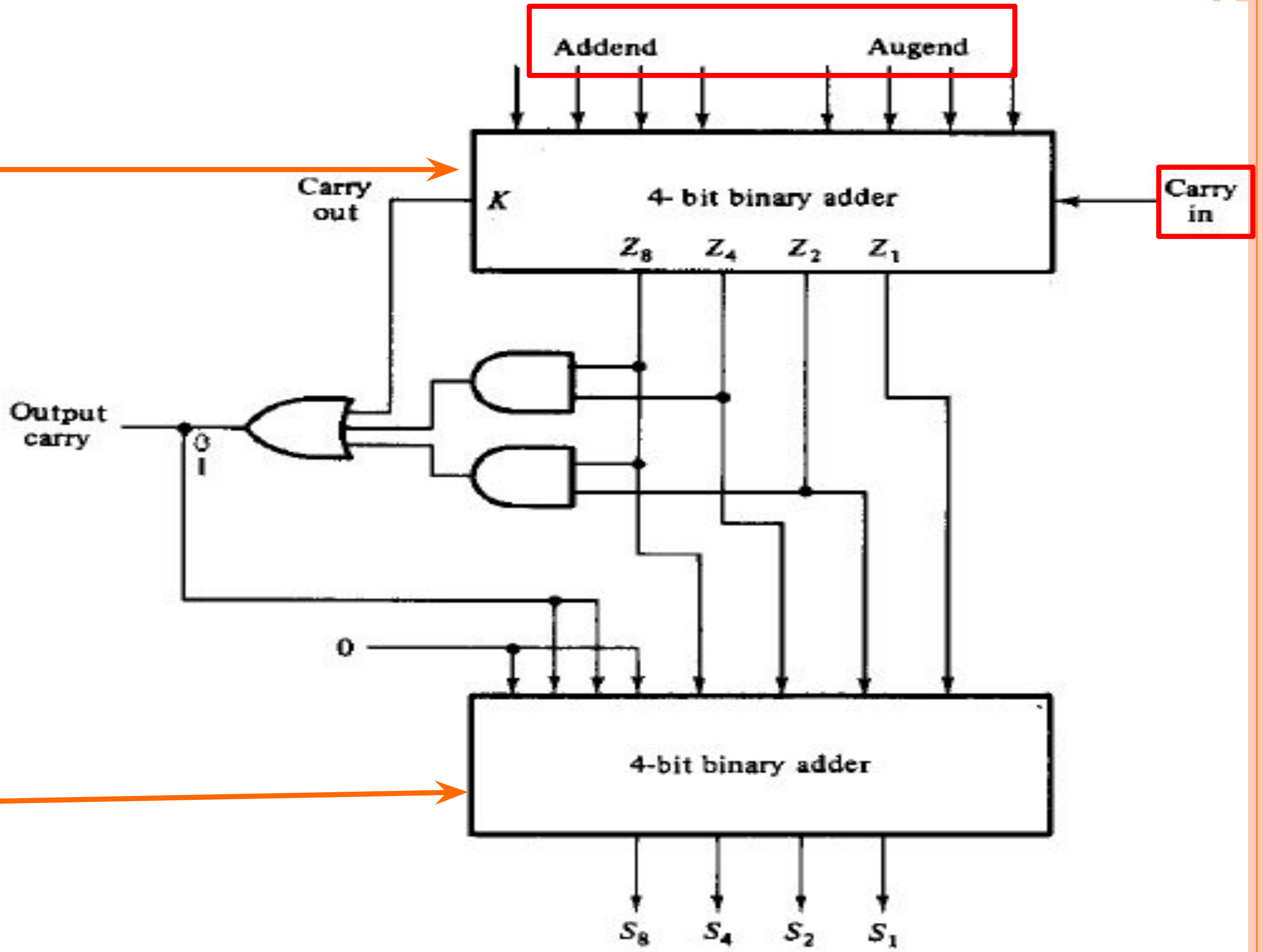
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1
0	1	0	0	0	0	1	0	0	0
0	1	0	0	1	0	1	0	0	1
0	1	0	1	0	1	0	0	0	0
0	1	0	1	1	1	0	0	0	1
0	1	1	0	0	1	0	0	1	0
0	1	1	0	1	1	0	0	1	1
0	1	1	1	0	1	0	1	0	0
0	1	1	1	1	1	0	1	0	1
1	0	0	0	0	1	0	1	1	0
1	0	0	0	1	1	0	1	1	1
1	0	0	1	0	1	1	0	0	0
1	0	0	1	1	1	1	0	0	1

- To distinguish them form the value 1000 and 1001 (also have a 1 in position Z8), we specify, either Z_4 or Z_2 must have a 1

$$C = K + Z_8Z_4 + Z_8Z_2$$

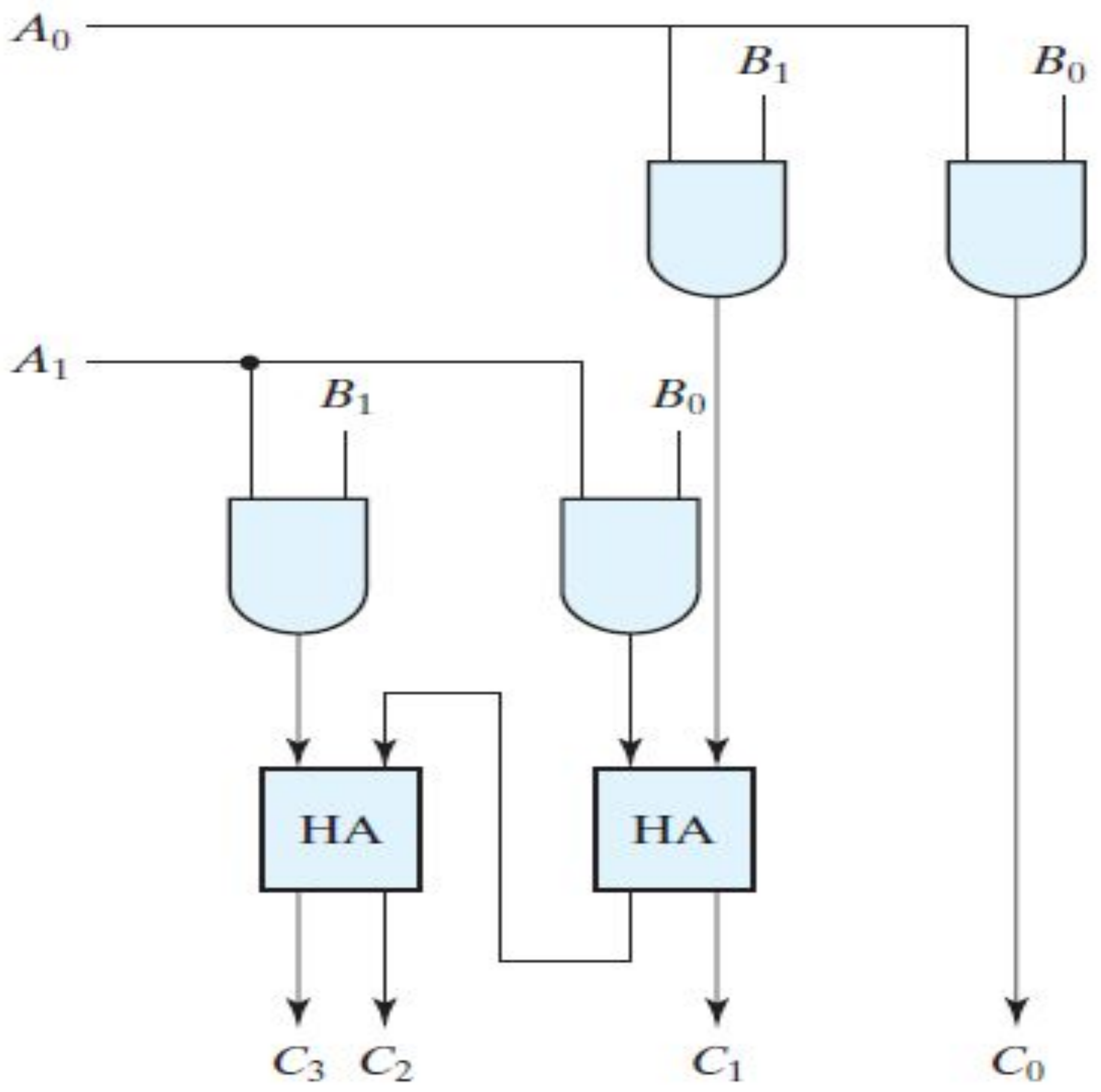
BCD ADDER

- When C = K + Z_8Z_4 + Z_8Z_2 ,it is necessary to add 110 to the binary sum and provide an output carry for the next stage
- To add 0110 to the binary sum, we use a second 4-bit binary adder
- First the **augend, added** and the **input carry** are added in the top 4-bit binary adder to produce the binary sum
- When output **carry=0 nothing added**
- When output **carry=1,binary 0110 is added** to the binary sum through the bottom 4-bit binary adder



2- BIT by 2-BIT BINARY MULTIPLIER

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



MAGNITUDE COMPARATOR

- **Magnitude Comparator:** A *magnitude comparator* is a combinational circuit that compares two numbers A and B, and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicate whether A=B , A>B , or A<B
- The circuit for comparing two n-bit numbers has 2^{2n} entries in the truth table and becomes too cumbersome even with n=3.
- So, we will develop an algorithm to design a comparator

MAGNITUDE COMPARATOR

- We will consider two 4-bit numbers:

  $A = A_3A_2A_1A_0$

  $B = B_3B_2B_1B_0$

- The two numbers are equal if **all pairs** of significant digits are equal. If $A_3=B_3$ & $A_2=B_2$ & $A_1=B_1$ & $A_0=B_0$
- So the equality relation of each pair of bits will be

  $\mathbf{x_i = A_iB_i+A_i'+B_i'}$ **,** where $x_i=1$ only if the pair of bits in position i are equal
- If all pair of bits of A and B are equal which means, if $X_3X_2X_1X_0$ all are equal 1, we can say A=B

MAGNITUDE COMPARATOR

- To determine if A>B or A<B, we inspect relative magnitudes of significant bits starting from the MSB
- If these two bits are equal, we compare the next lower significant pair of bits
- This comparison continues until a pair of unequal digits is reached
    - If the corresponding digit of A=1 and that of B=0, we conclude A>B
    - If the corresponding digit of A=0 and that of B=1, we conclude A<B
- The boolean function for the above sequential comparison will be:

A_3	A_2	A_1	A_0
B_3	B_2	B_1	B_0

1	1	1	1
1	1	0	1

1	1	0	1
1	1	1	1

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

MAGNITUDE COMPARATOR

If $A_3B_3' = 1$, A>B

A_3	A_2	A_1	A_0
B_3	B_2	B_1	B_0

1	1	1	1
0	1	0	1

If $X_3A_2B_2' = 1$, A>B

1	1	0	1
1	0	1	1

If $X_3X_2A_1B_1' = 1$, A>B

1	1	1	1
1	1	0	1

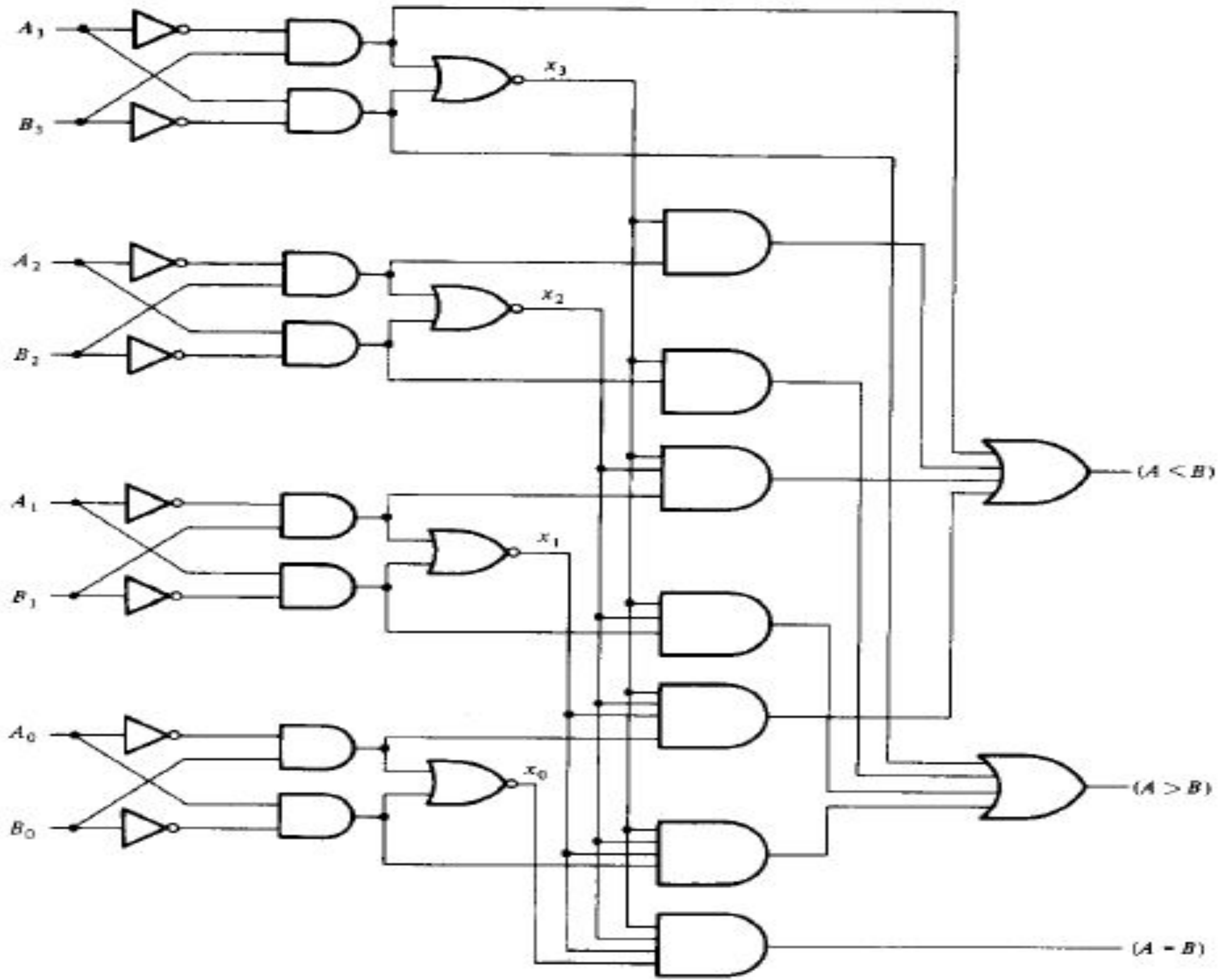
If $X_3X_2X_1A_0B_0' = 1$, A>B

1	1	1	1
1	1	1	0

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

MAGNITUDE COMPARATOR



MAGNITUDE COMPARATOR

- **$x_i = A_iB_i + A_i' + B_i'$**
- **$(A_0'B_0 + A_0B_0')' = (A_0 \text{ xor } B_0)'$**

  = A0 xnor B0

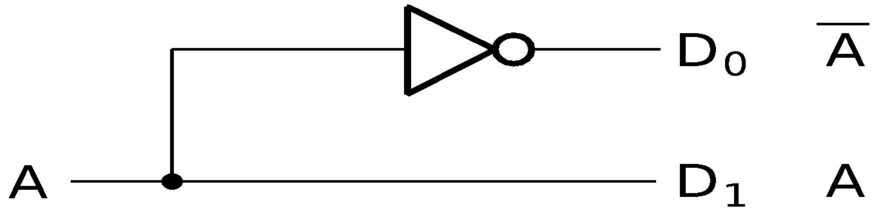
  $= A_0B_0 + A_0'B_0'$

DECODER

- A *decoder* is a combinational circuit that converts binary information from *n-input* lines to a maximum *2^n unique output lines.*
- If the *n-bit* decoded information has unused or don't care combinations, the decoder output will have fewer than 2^n outputs
- This decoders are called *n-to-m-line* decoders, where $m \leq 2^n$
- 1-to-2-Line Decoder -

A	**D_0**	**D_1**
0	1	0
1	0	1

(a)

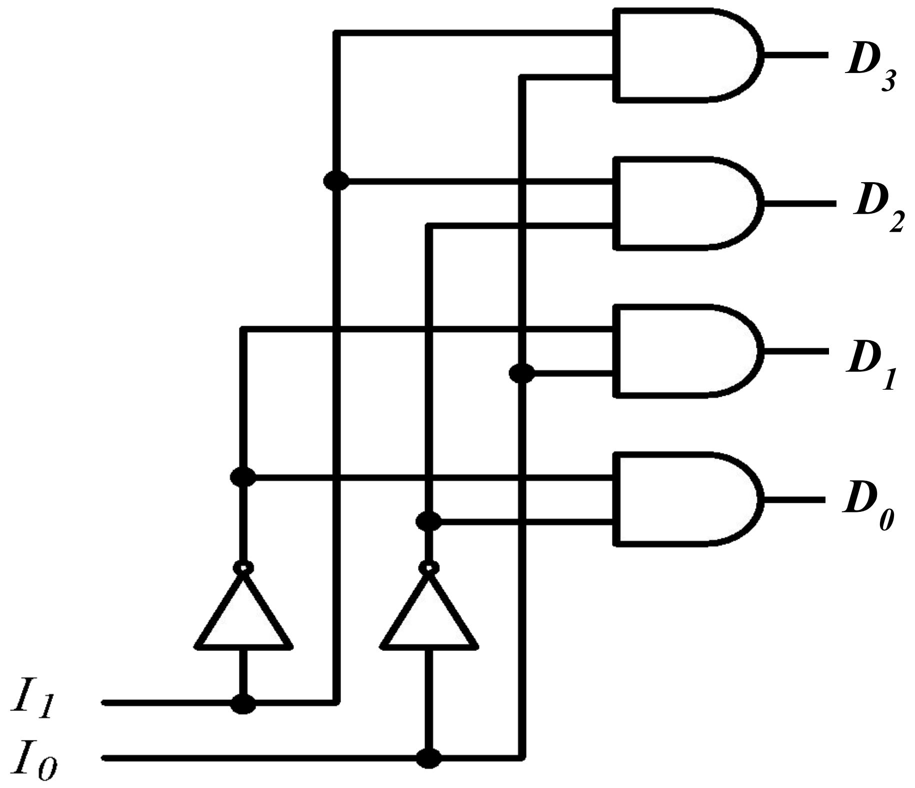


(b)

2-to-4 Line Decoder

- The 2 inputs are decoded into 4 outputs
- Each output represents one of the minterms of the 2-input variables
- 2 intverters provide the complement of the inputs
- Each one of the 4 AND gates generate one of the minterms

A B	D_3	D_2	D_1	D_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

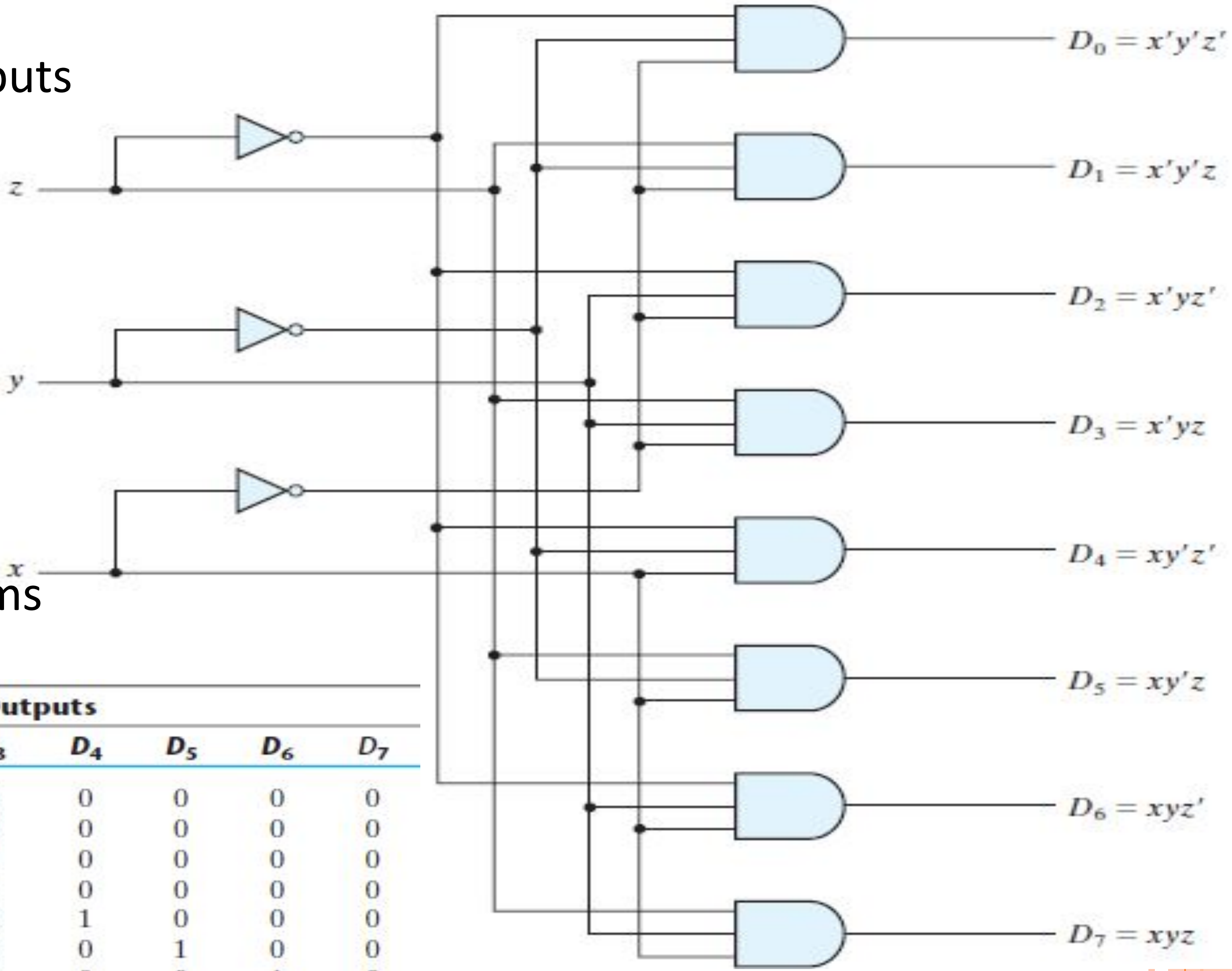


$$D_3 = AB \qquad D_1 = A'B$$

$$D_2 = AB' \qquad D_0 = A'B'$$

3-to-8 Line Decoder

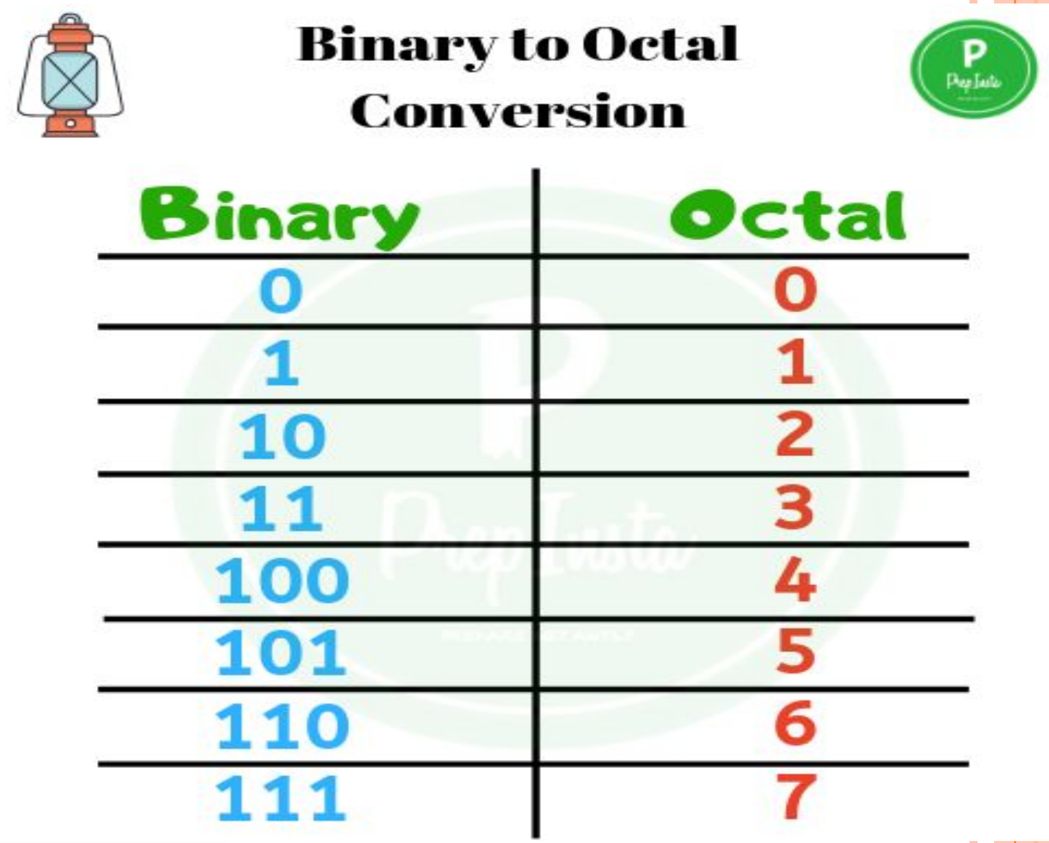
- 3 inputs are decoded into 8 outputs
- Each output represents one of the minterms of the 3-input variables
- 3 intverters provide the complement of the inputs
- Each one of the 8 AND gates generate one of the minterms



Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_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

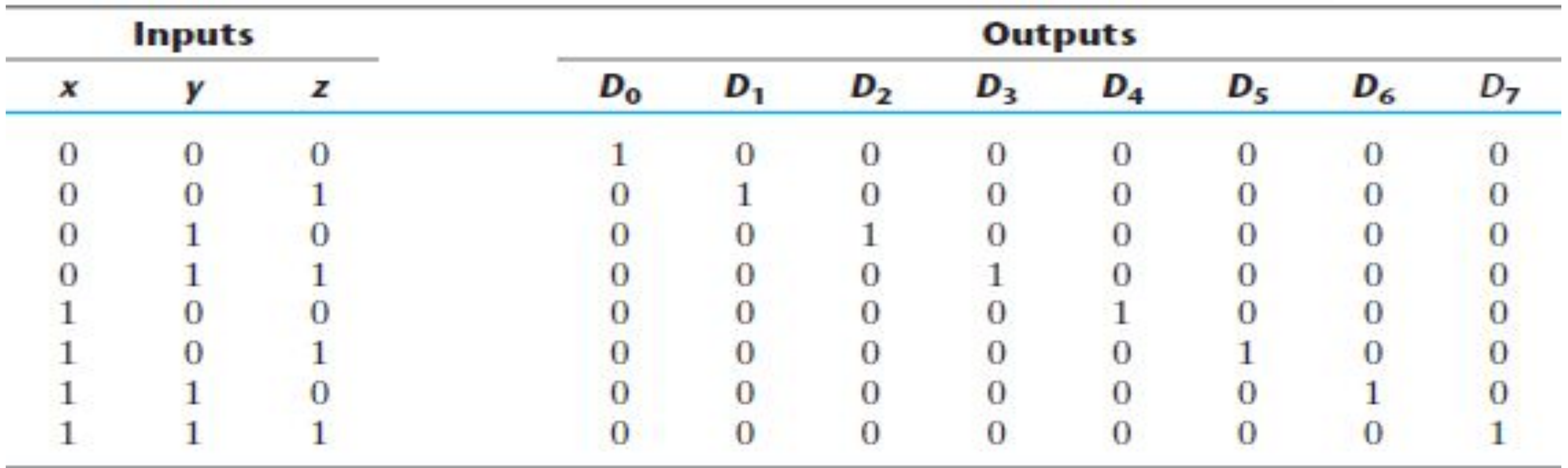
3-to-8 Line Decoder

- A particular application of this decoder is binary-to-octal conversion.
- The input variables represent a binary number
- The outputs represent the eight digits of a number in octal number system



Binary to Octal Conversion

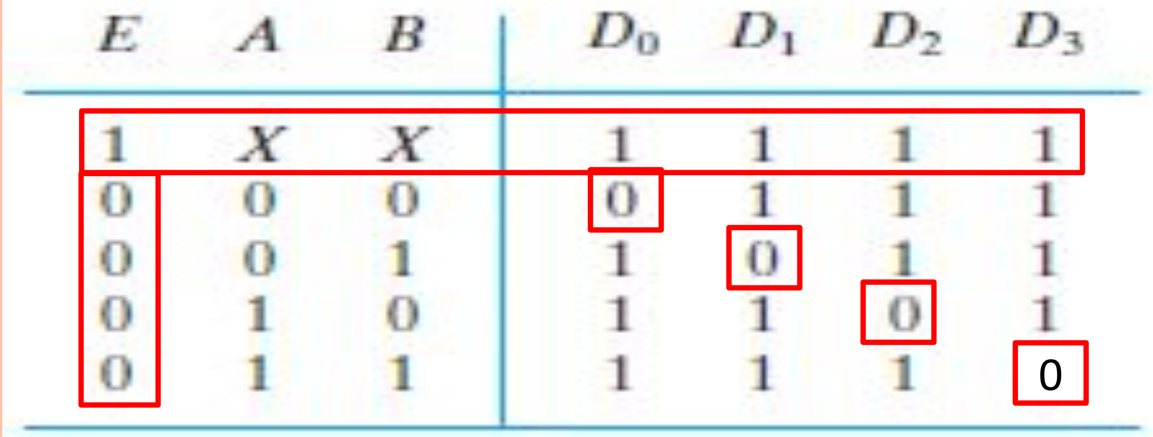
Binary	Octal
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7



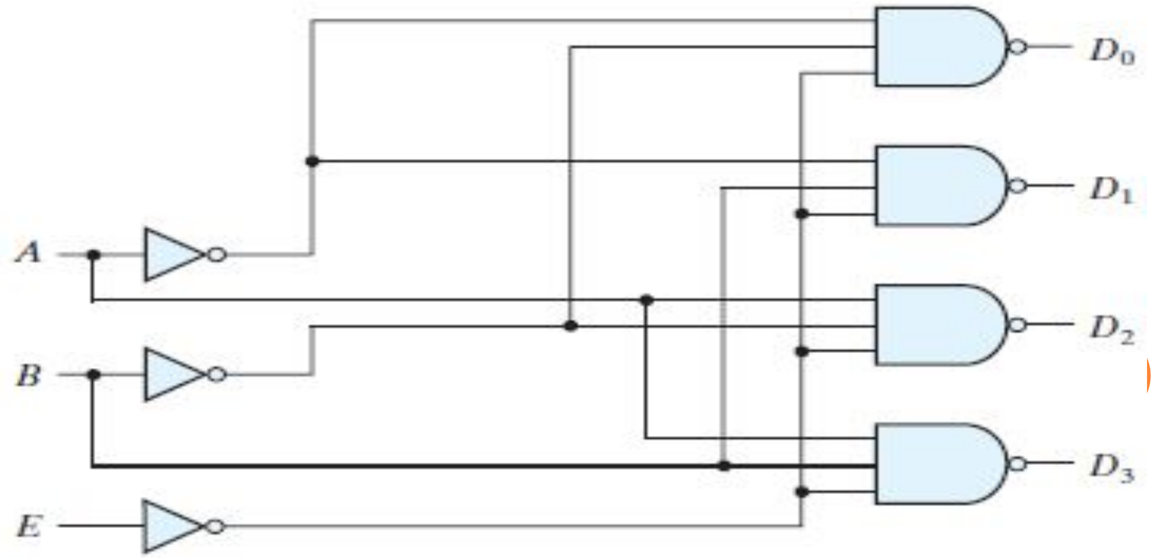
Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

2-to-4 Line Decoder with enable input

- Some decoders are constructed with **NAND** gates
- May include one or more ***Enable*** inputs to control the circuit operation
- Circuit operates with **complemented output**
- and **complement enable input**
- The decoder is **enabled** when E=0
- Circuit is disabled when E=1
- When circuit is disabled, none of the outputs are equal to 0 and none of the minterms are selected
- At any given time, only one output is equal to 0, other outputs are 1

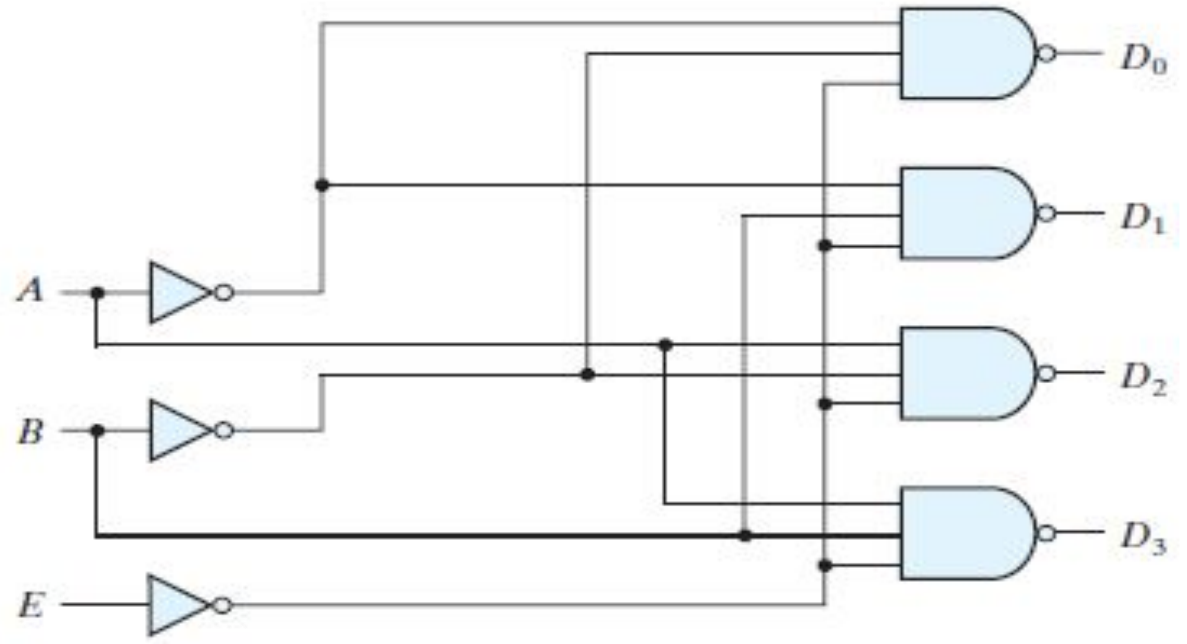


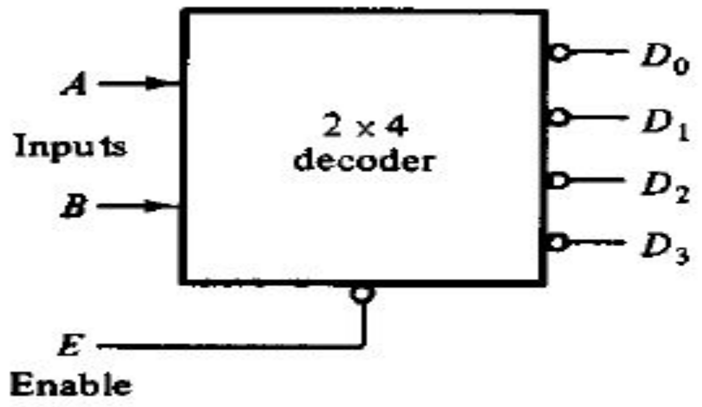
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0



Decoder

- A decoder can operate with complemented or uncomplemented output
- Enable input may be activated with a 0 or a 1 signal
- Some decoders may have two or more enable inputs
- Block diagram of the 2-to-4 line decoder -





(a) Decoder with enable

COMBINATIONAL LOGIC IMPLEMENTATION

- A decoder produces *2^n* minterms of *n-inputs*
- Any boolean function can be expressed in sum of minterms form
- So, a decoder can be used to generate minterms, and an external OR gate to form the sum

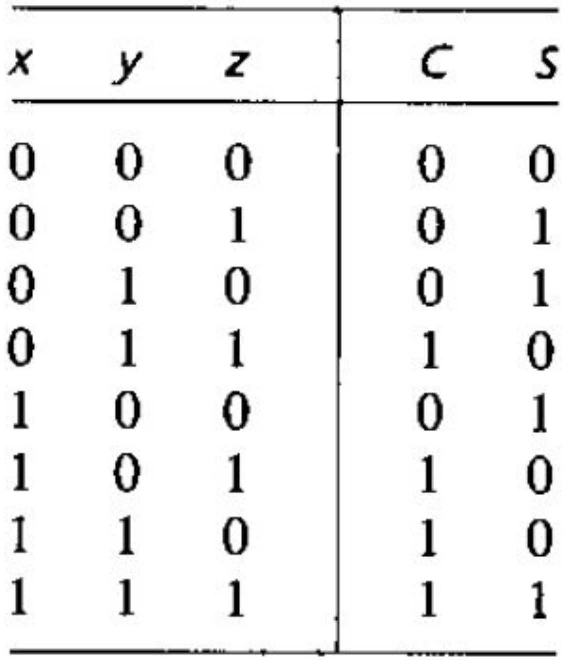
- Any combinational circuit can be implemented with a decoder and OR gates
- Implementing a combinational circuit with a decoder and OR gates requires the **boolean function of the circuit be expressed in sum of minterms**

IMPLEMENTING FULL ADDER USING DECODER

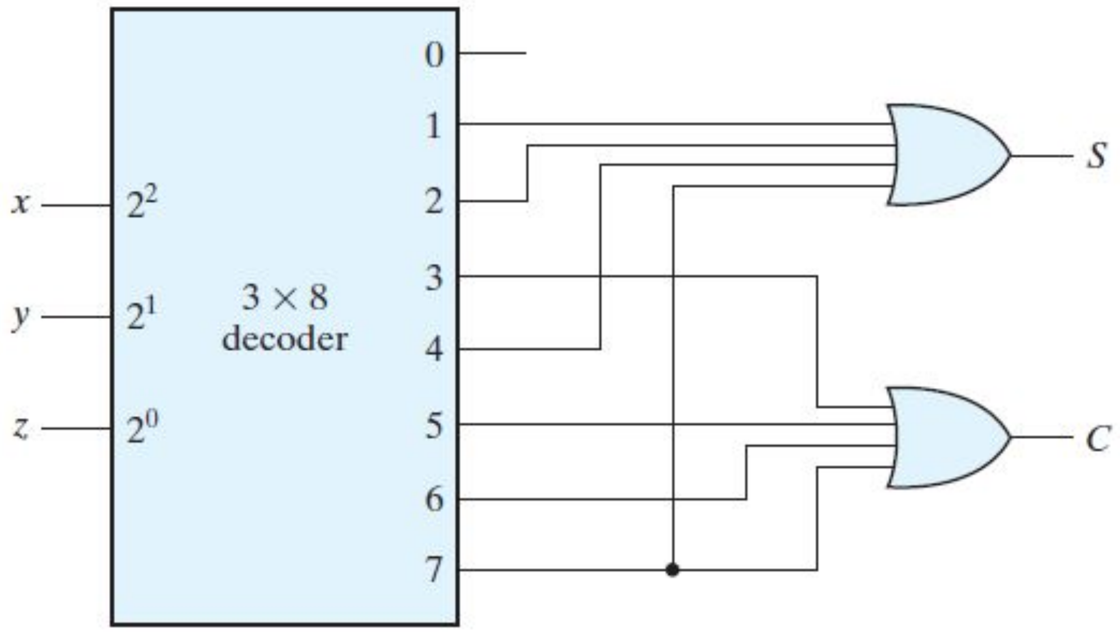
- From the table of the full-adder, function for this circuit in sum of minterms:

$$S = X'Y'Z + X'YZ' + XY'Z' + XYZ \quad \text{or} \quad S(x, y, z) = \sum(1, 2, 4, 7)$$
$$C = X'YZ + XY'Z + XYZ' + XYZ \quad \text{or} \quad C(x, y, z) = \sum(3, 5, 6, 7)$$

- Full-adder circuit has 3 inputs and total 8 outputs, we need **3-to-8-line decoder**
- Decoder generates the 8 minterms for x,y,z
- The OR gate for **output S** forms the sum of minterms **1,2,4 and 7**
- The OR gate for **output C** forms the sum of minterms **3,5,6,and 7**



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

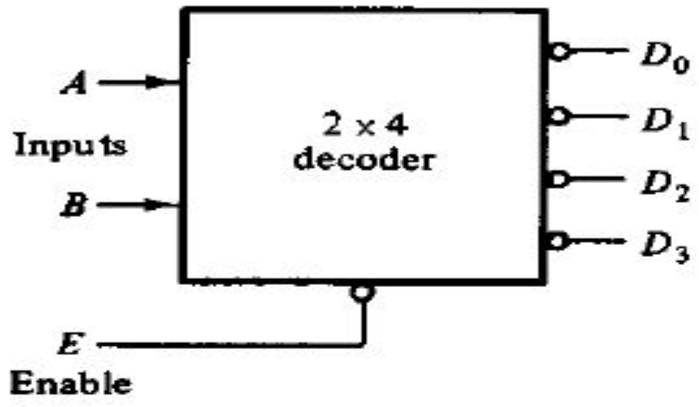


DEMULTIPLEXER

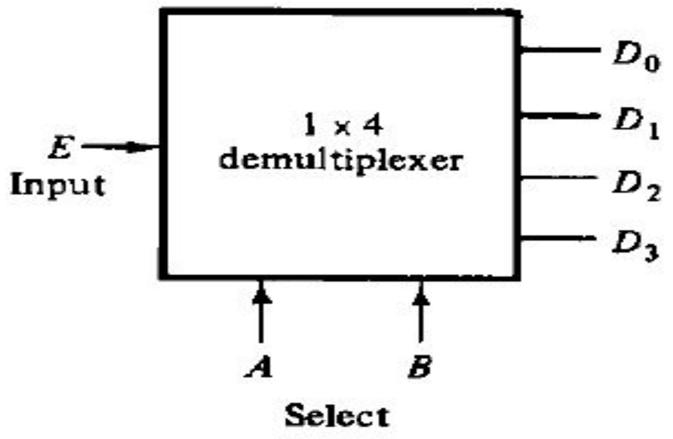
- **Demultiplexer:** A *demultiplexer* is a circuit that **receives information** from **a single line** and **transmits** this information it to **one of 2^n possible output lines.**

- The selection of a specific output is controlled by the bit combination of n selection lines.

- **The decoder can function as 1-to-4-line demultiplexer** when **E** is taken as a **data input line** and **A and B** are taken as the **selection inputs**.



(a) Decoder with enable



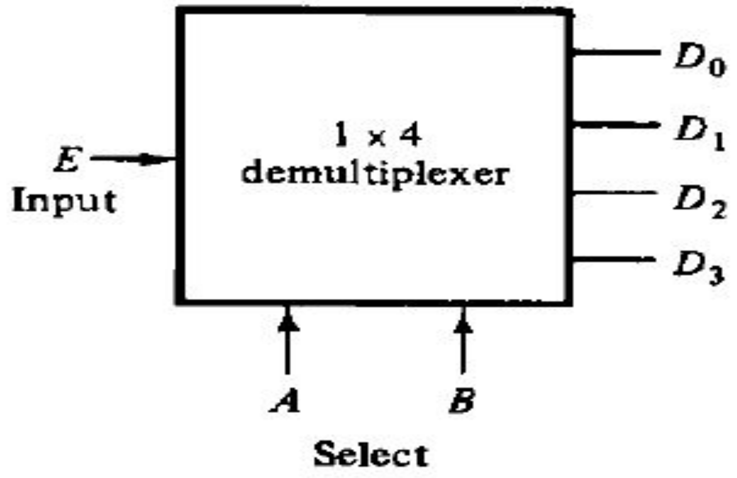
(b) Demultiplexer

DEMULTIPLEXER

The single input variable **E** has a path to **all four outputs**, but the input information is **directed** to only **one of the output lines**, as specified by the binary combination of the two selection lines A and B

Example, if the selection lines **AB = 10**, output **D_2** will be the **same as the input value E** , while all other outputs are maintained at 1.

As decoder and demultiplexer operations are obtained from the same circuit, a decoder with an enable input is referred to as a decoder – demultiplexer

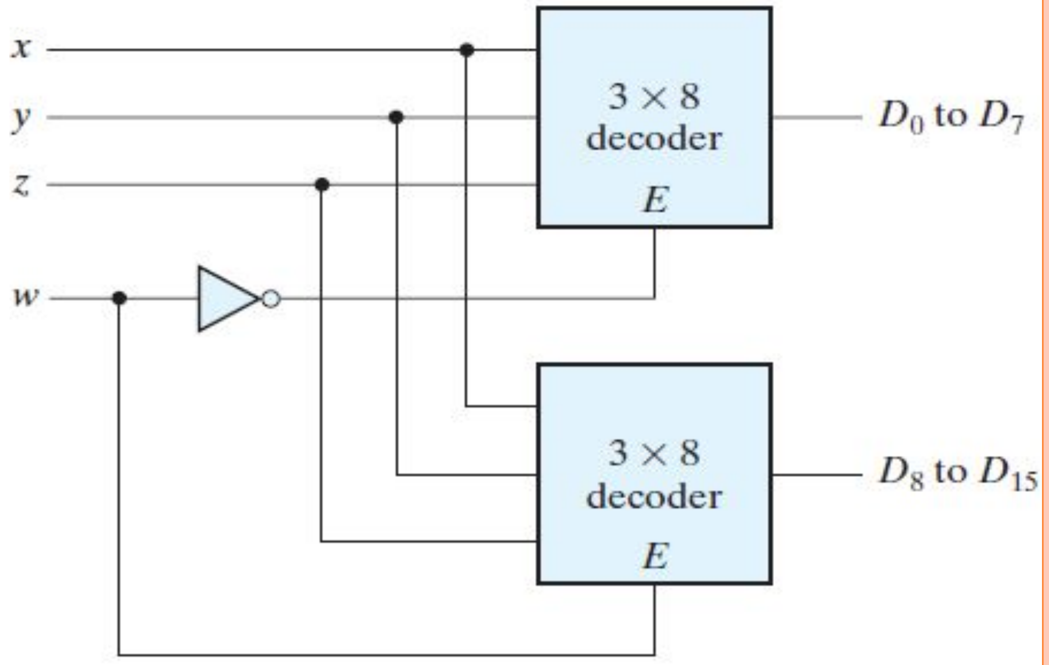


(b) Demultiplexer

DECODER EXPANSION

Decoder/ Demultiplexer can be connected together to form a larger decoder circuit

- Example, **two 3-to-8-line decoders** with enable inputs connected to form **a 4-to-16-line decoder**
- When ***w* = 0**, *the top decoder is enabled and the other is disabled.*
- The bottom decoder outputs are all 0's, and the top eight outputs generate minterms 0000 to 0111.
- When ***w* = 1**, *the enable conditions are reversed: The bottom decoder* outputs generate minterms 1000 to 1111, while the outputs of the top decoder are all 0's.



x
y
z
w
3×8 decoder
E
D_0 to D_7
3×8 decoder
E
D_8 to D_{15}