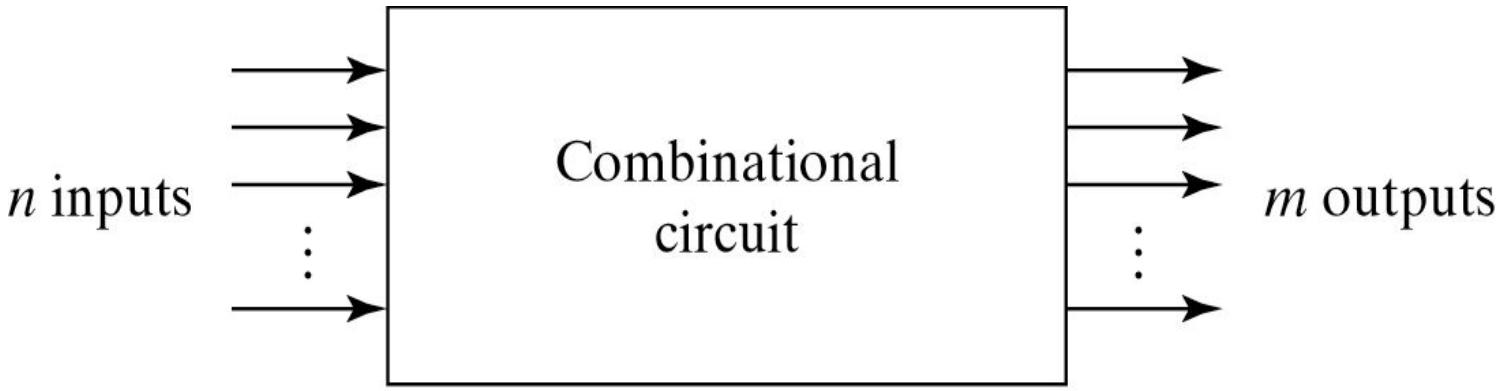
COMBINATIONAL LOGIC


n inputs
Combinational
circuit
m outputs

Half Adder

Combinational Circuit Which Performs Addition
Of Two Binary Digits

Truth Table for Half Adder

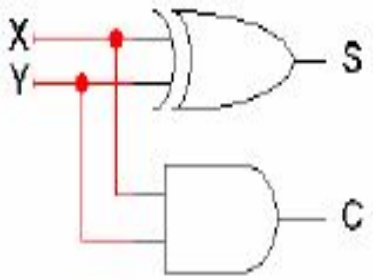
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

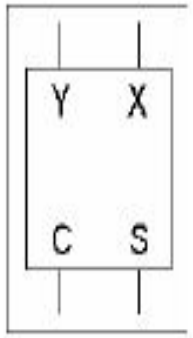
Equations for Half Adder

S(Sum)=AB’+A’B

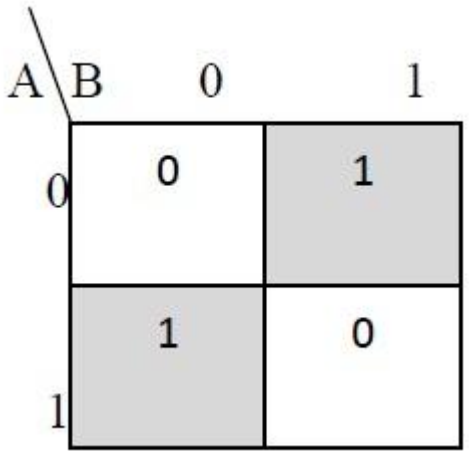
S=A EXOR B

C(Carry)=A.B

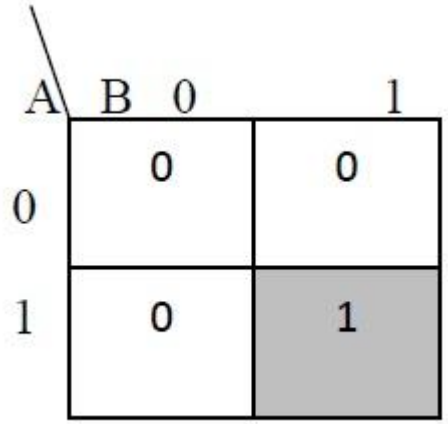




K Map for Sum



K Map for Carry



Equations for Half Adder

S(Sum)=AB’+A’B

S=A EXOR B

C(Carry)=A.B

Full Adder

Combinational Circuit Which Performs Addition
Of Three Input Bits and gives Two Outputs

Truth Table for Full Adder

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Equations for Full Adder

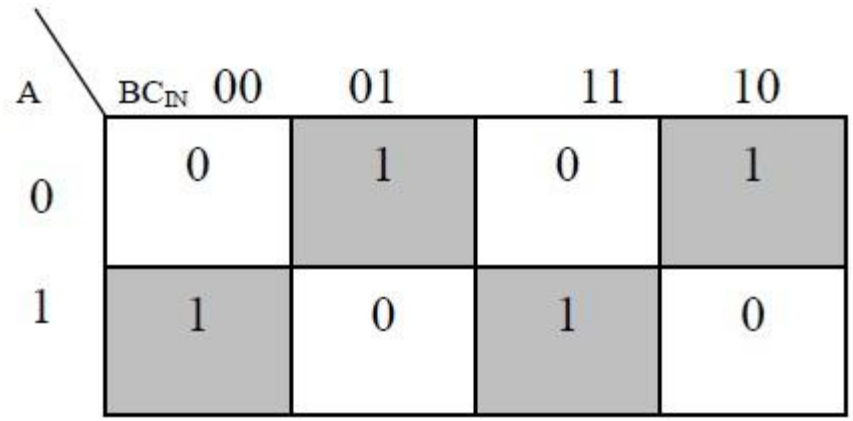
S(Sum)=A'B'Cin+AB'Cin'+A'BCin'+ABCin

S=A EXOR B EXOR Cin

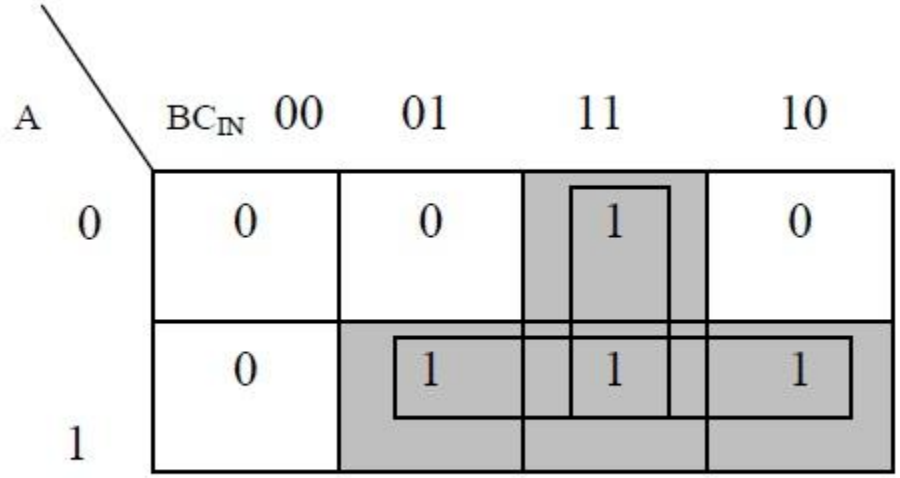
C(Carry)=A.Cin+ A.B + B.Cin

Taking
x=A
y=B
z=Cin

KMAP for SUM



KMAP for CARRY



Equations for Full Adder

S(Sum)=A'B'Cin+AB'Cin'+A'BCin'+ABCin

S=A EXOR B EXOR Cin

C(Carry)=A.Cin+ A.B + B.Cin

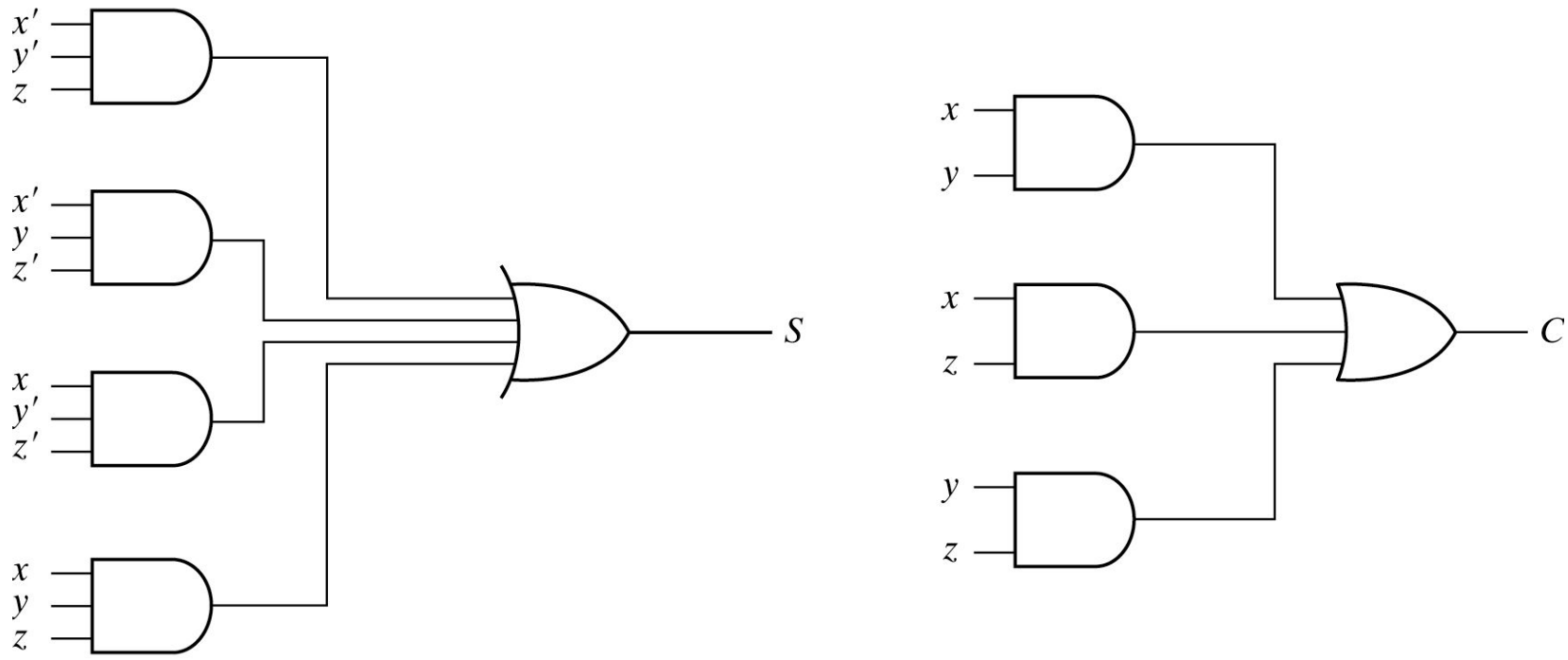
Taking

x=A

y=B

z=Cin

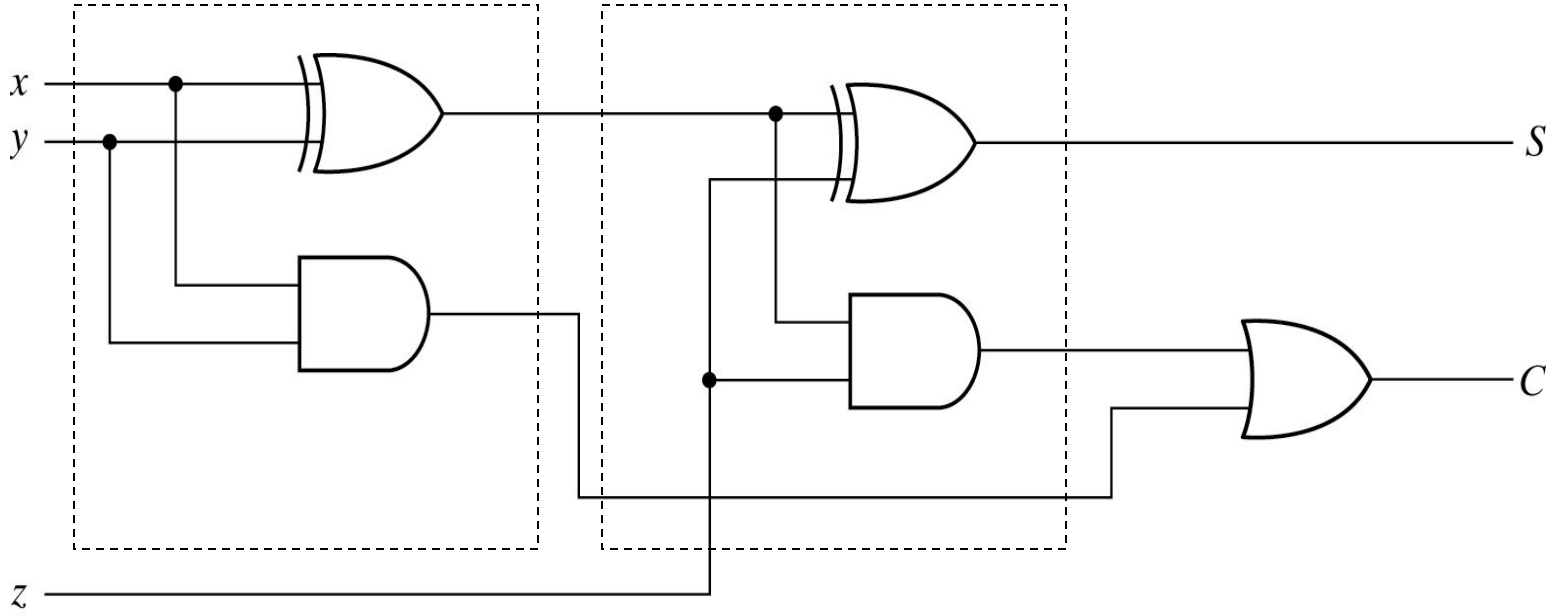
Implementation of Full Adder using AND-OR Gate Network



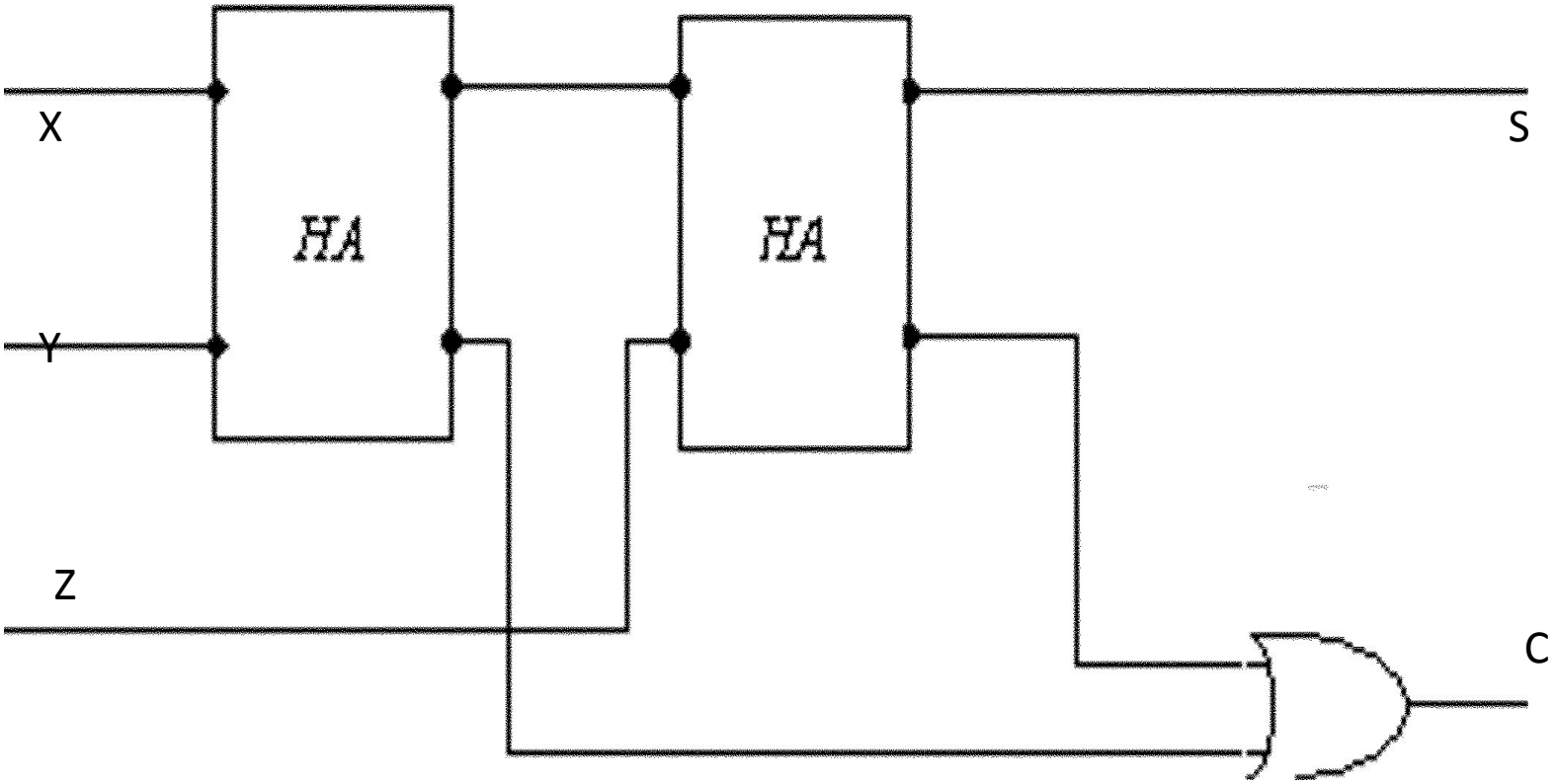
Implementation of Full Adder using Two Half Adders …

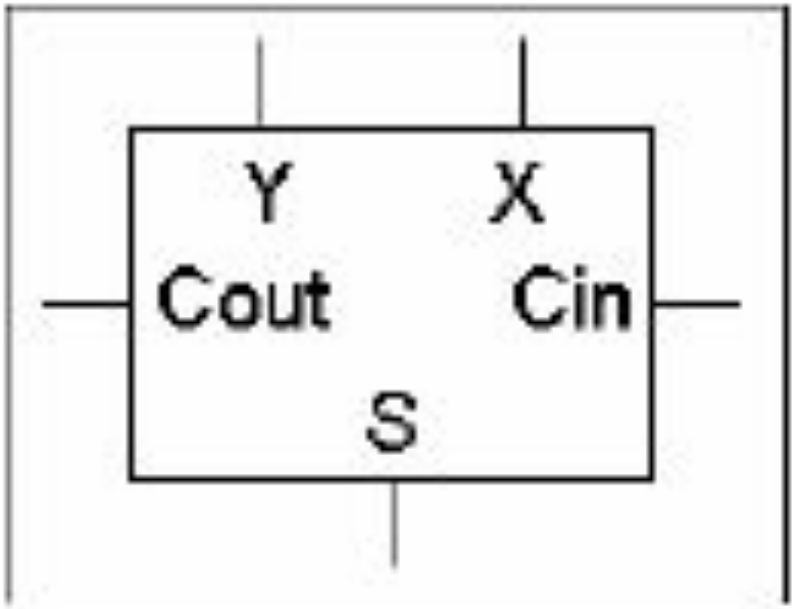
S=XY'Z' +X'YZ' + XYZ + X'Y'Z
= Z' (XY'+X'Y) + Z (XY + X'Y')
= Z' (XY' + X'Y) + Z (XY' + X'Y)'
= Z (X ⊕ Y) ⊕

C=XZ+XY+YZ
CONVERTING IT TO CANONICAL FORM GIVES
C= XY'Z + X'YZ + XYZ'+XYZ
= XY'Z + X'YZ + XY(Z'+Z)
= XY'Z + X'YZ + XY
= Z(XY' + X'Y) + XY
= Z (X ⊕ Y) + XY



Implementation of Full Adder using Two Half Adders




Y
X
Cout
Cin
S

Half Subtractor

Combinational Circuit that subtracts two bits and produces their difference and borrow

Truth Table for Half Subtractor

A	B	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Truth Table for Half Subtractor

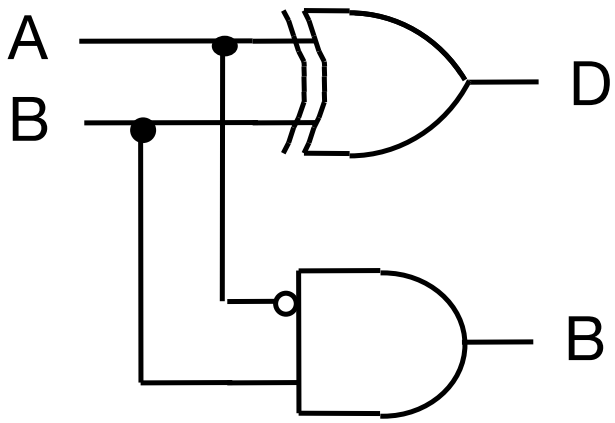
A	B	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Equations for Half Subtractor

D(Difference)=AB’+A’B

D=A EXOR B

B(Borrow)=A’.B



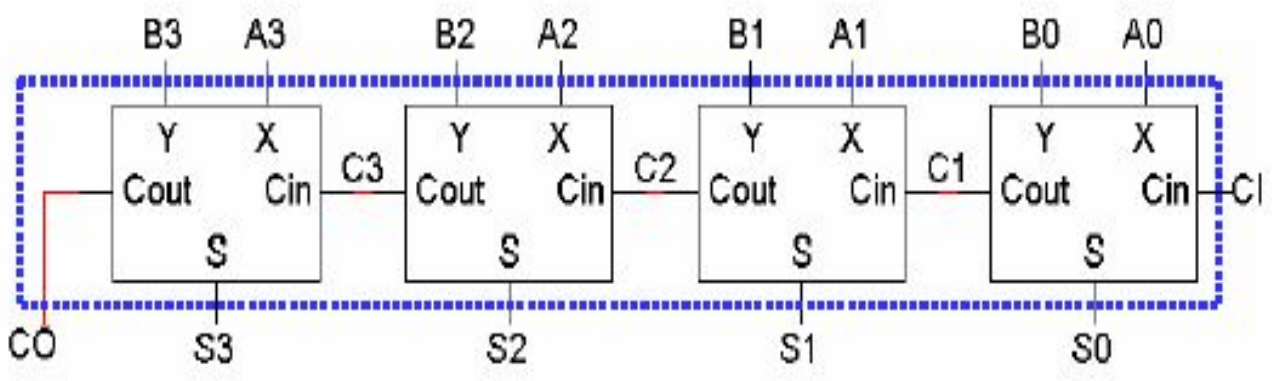
Full Subtractor

Combinational Circuit Which Performs Subtraction involving Three Bits, i.e. A,B and Bin(Borrow from previous stage) and gives Two Outputs D(Diff) and B(Borrow)

Truth Table for Full Subtractor

A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Four Bit Binary Adder



BCD ADDER

Adds two BCD digits using rules of Binary addition and produces a BCD digit.
A BCD digit cannot be greater than 9

If the sum <=9 and carry=0 then
Sum is correct, in true BCD form

If sum is invalid BCD i.e. >9 or carry =1 then
The Sum is wrong and needs correction.

Correction by adding 6 (0110) to the sum

Truthtable FOR SUM>9

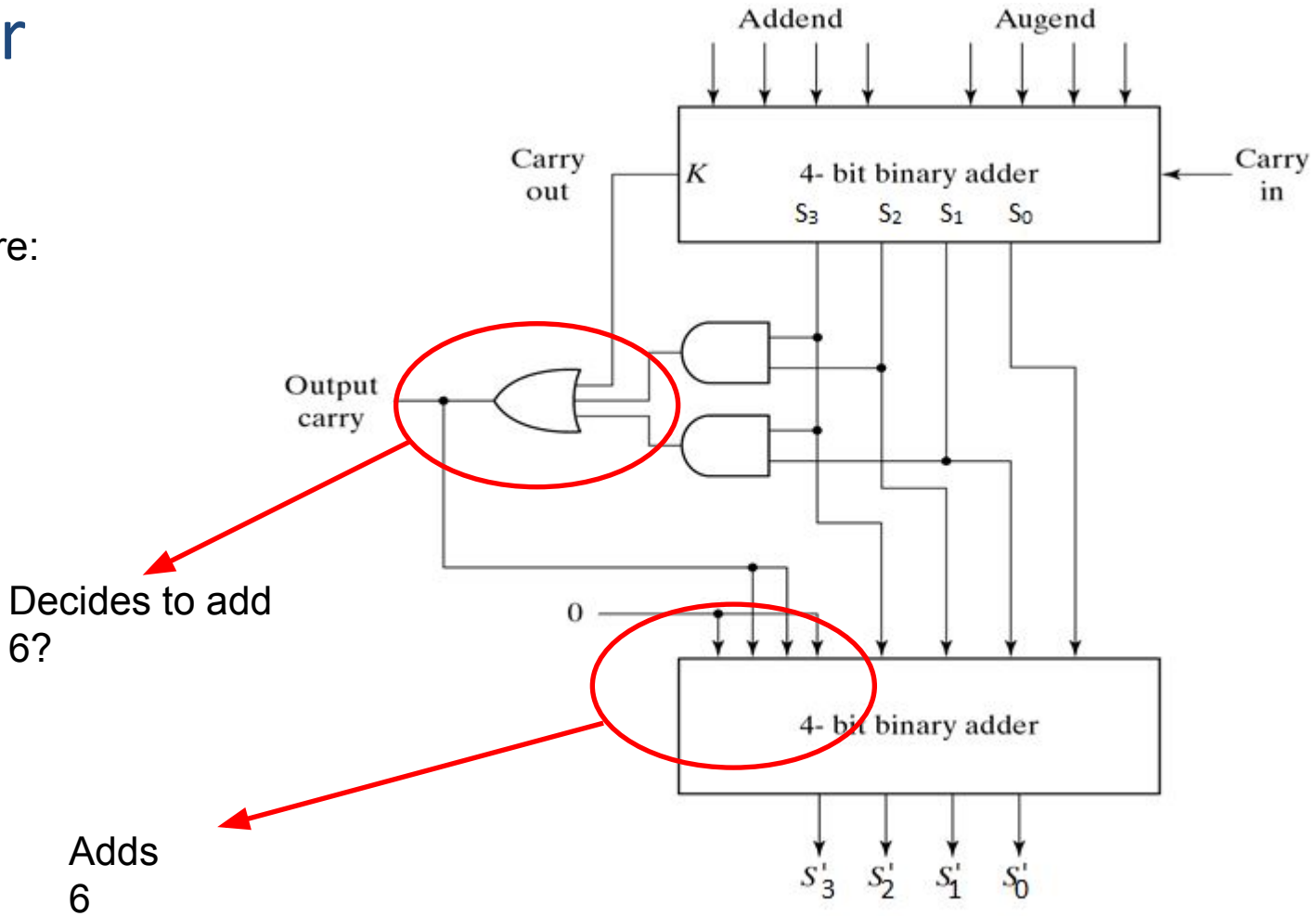
Kmap gives:

$Y=S_3 S_2 + S_3 S_1$

S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

BCD adder

Numbers that need correction (add 6) are:
01010 (10)
01011 (11)
01100 (12)
01101 (13)
01110 (14)
01111 (15)
10000 (16)
10001 (17)
10010 (18)
10011 (19)

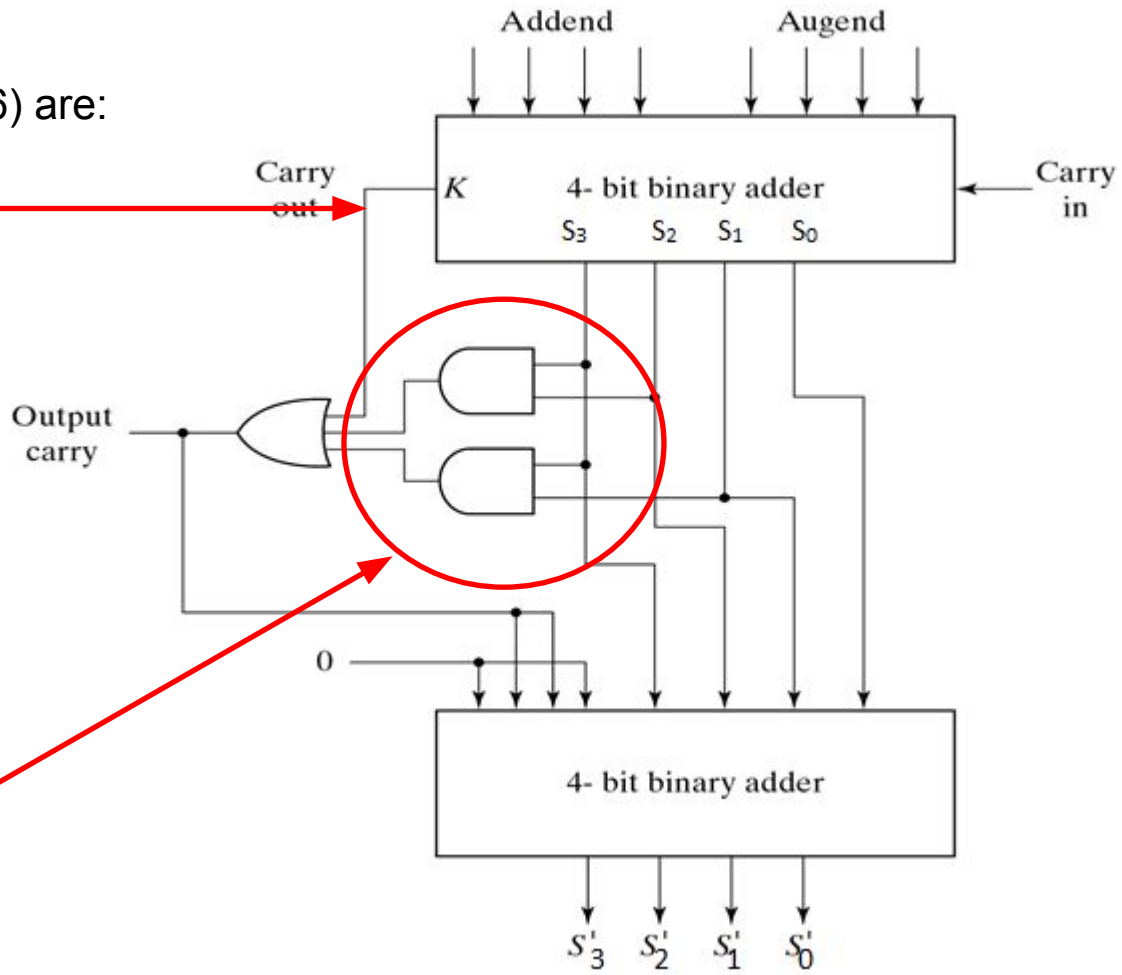


BCD adder

Numbers that need correction (add 6) are:

K	S_3	S_2	S_1	S_0	
0	1	0	1	0	(10)
0	1	0	1	1	(11)
0	1	1	0	0	(12)
0	1	1	0	1	(13)
0	1	1	1	0	(14)
0	1	1	1	1	(15)
1	0	0	0	0	(16)
1	0	0	0	1	(17)
1	0	0	1	0	(18)
1	0	0	1	1	(19)

$C = K + S_3 S_2 + S_3 S_1$



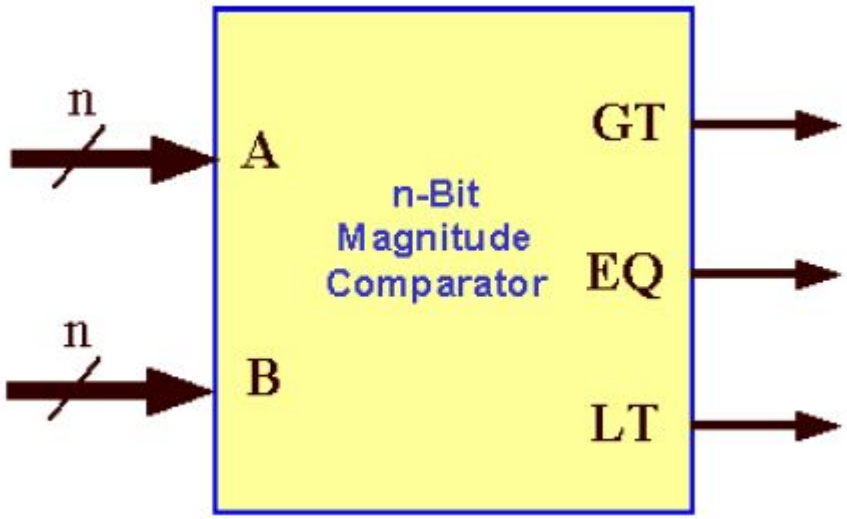
Magnitude Comparator

- It is a combinational circuit that compares to numbers and determines their relative magnitude
- The output of comparator is usually 3 binary variables indicating: A>B

  A=B

  A<B
- For example to design a comparator for 2 bit binary numbers A (A1A0) and B (B1B0) we do the following steps:

N Bit Magnitude Comparator



1 Bit Comparator Truth table

A	B	A>B	A<B	A=B
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

Truth table for 2-bit Magnitude Comparator

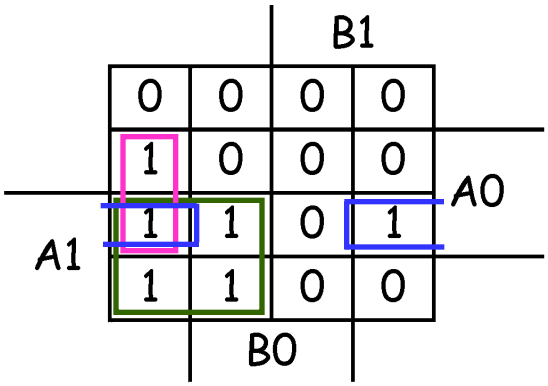
A1	A0	B1	B0	G	E	L
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

G for A>B
E for A+B
L for A<B

K Map for 2 Bit comparator

Let's use K-maps. There are *three* functions (each with the same inputs

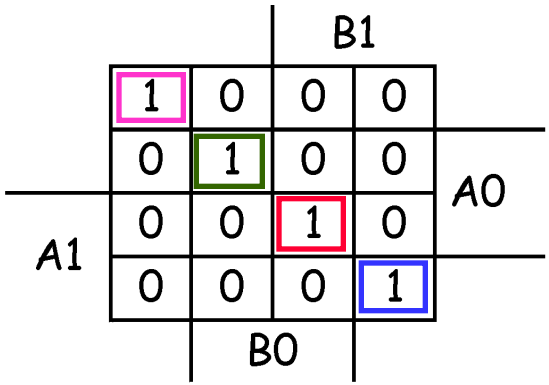
(A1 A0 B1 B0), so we need *three* K-maps



A1	A0	B1	B0	G	E	L
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
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	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

G(A1,A0,B1,B0) = A1 A0 B0' + A0 B1' B0' + A1 B1'

K Map for 2 Bit comparator

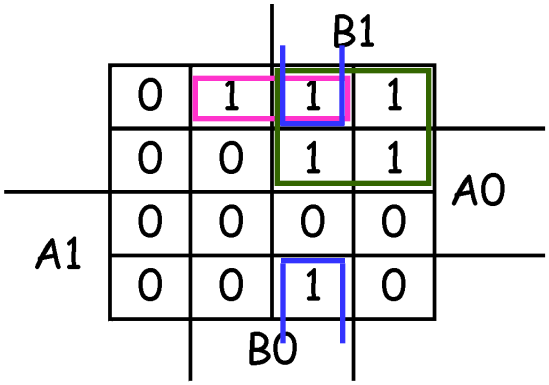


A1	A0	B1	B0	G	E	L
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

E(A1,A0,B1,B0) = A1' A0' B1' B0' + A1' A0 B1' B0 + A1 A0 B1 B0 + A1 A0' B1 B0'

= (A0 EXNOR B0) (A1 EXNOR B1)

K Map for 2 Bit comparator



A1	A0	B1	B0	G	E	L
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
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	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

L(A1, A0, B1, B0) = A1' A0' B0 + A0' B1 B0 + A1' B1

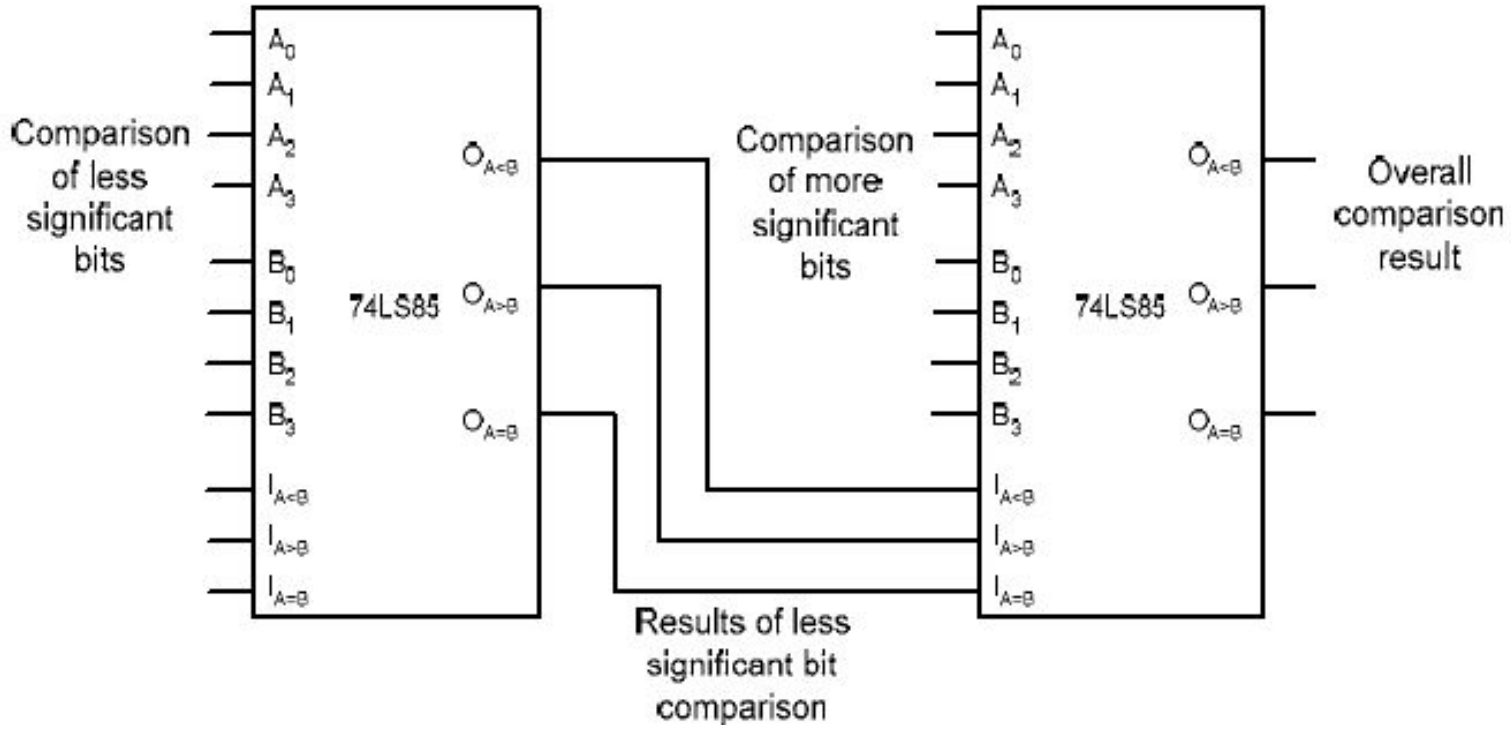
4 Bit Magnitude Comparator

Comparing Inputs				Cascaded Inputs			Outputs		
A3,B3	A2,B2	A1,B1	A0,B0	Ain>Bin	Ain<Bin	Ain=Bin	A>B	A<B	A=B
A3>B3	x	x	x	x	x	x	1	0	0
A3<B3	x	x	x	x	x	x	0	1	0
A3=B3	A2>B2	x	x	x	x	x	1	0	0
A3=B3	A2<B2	x	x	x	x	x	0	1	0
A3=B3	A2=B2	A1>B1	x	x	x	x	1	0	0
A3=B3	A2=B2	A1<B1	x	x	x	x	0	1	0
A3=B3	A2=B2	A1=B1	A0>B0	x	x	x	1	0	0
A3=B3	A2=B2	A1=B1	A0<B0	x	x	x	0	1	0
A3=B3	A2=B2	A1=B1	A0=B0	1	0	0	1	0	0
A3=B3	A2=B2	A1=B1	A0=B0	0	1	0	0	1	0
A3=B3	A2=B2	A1=B1	A0=B0	0	0	1	0	0	1

Only one of these three inputs can be active at a time

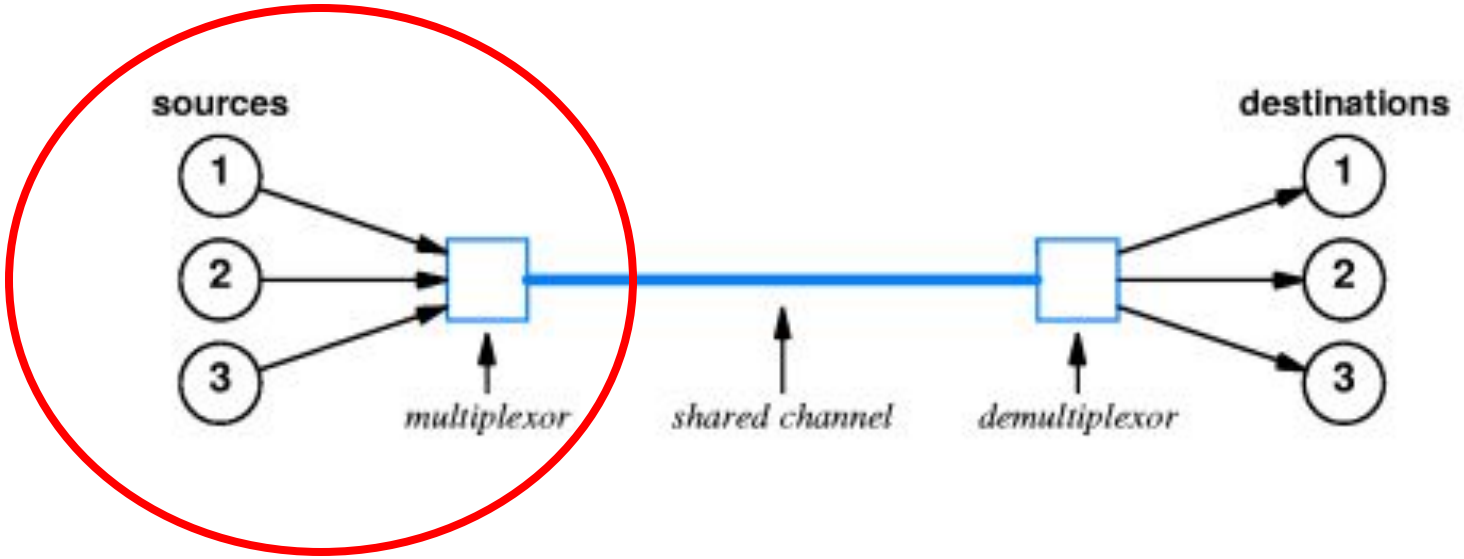
From less significant-bits

Cascadable 4-Bit Magnitude Comparator

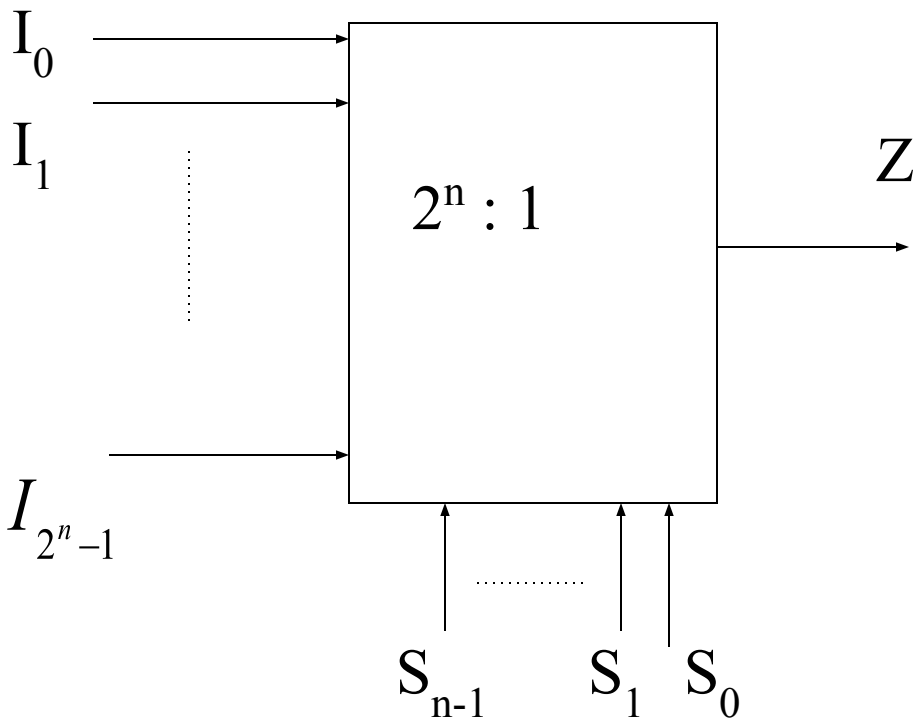


- One thing to notice is how the IC 74LS85 uses the cascaded inputs.
  - They can only affect the output if the current set of 4 bits are equal. This design forces us to compare the lower bits first and pass the outputs of the lower order comparison to the next comparator.

MULTIPLEXERS AND DEMULTIPLEXERS

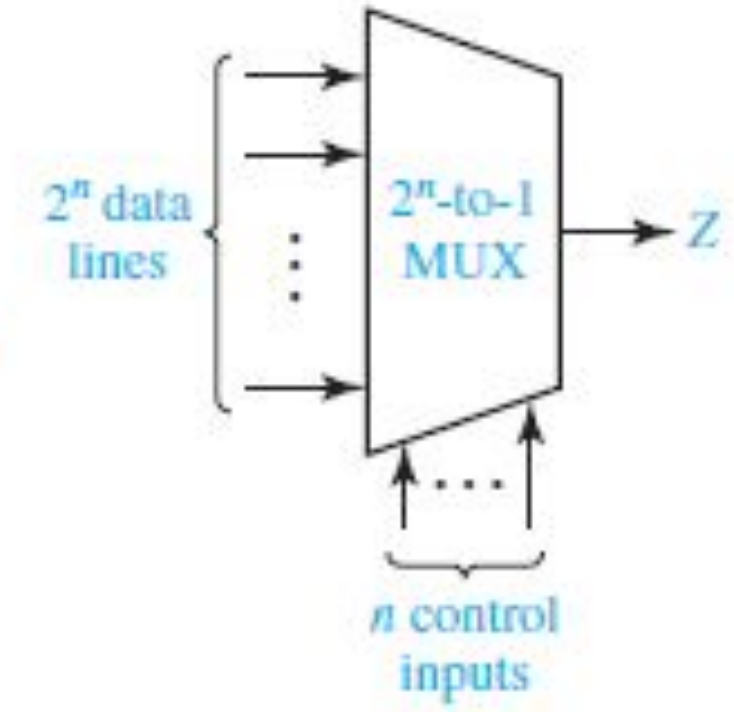


Multiplexing allows one to select one of the many possible sources.



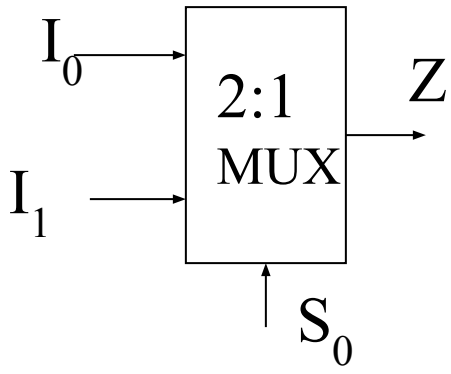
In general for a 2^n:1 MUX with control signals/inputs $S_{n-1} \cdots S_1 S_0$ & "data" inputs $I_0, \cdots, I_{2^n-1}$,
$Z = I_i$ when $S_{n-1} \cdots S_1 S_0$ combination represents #i in binary.

Multiplexers

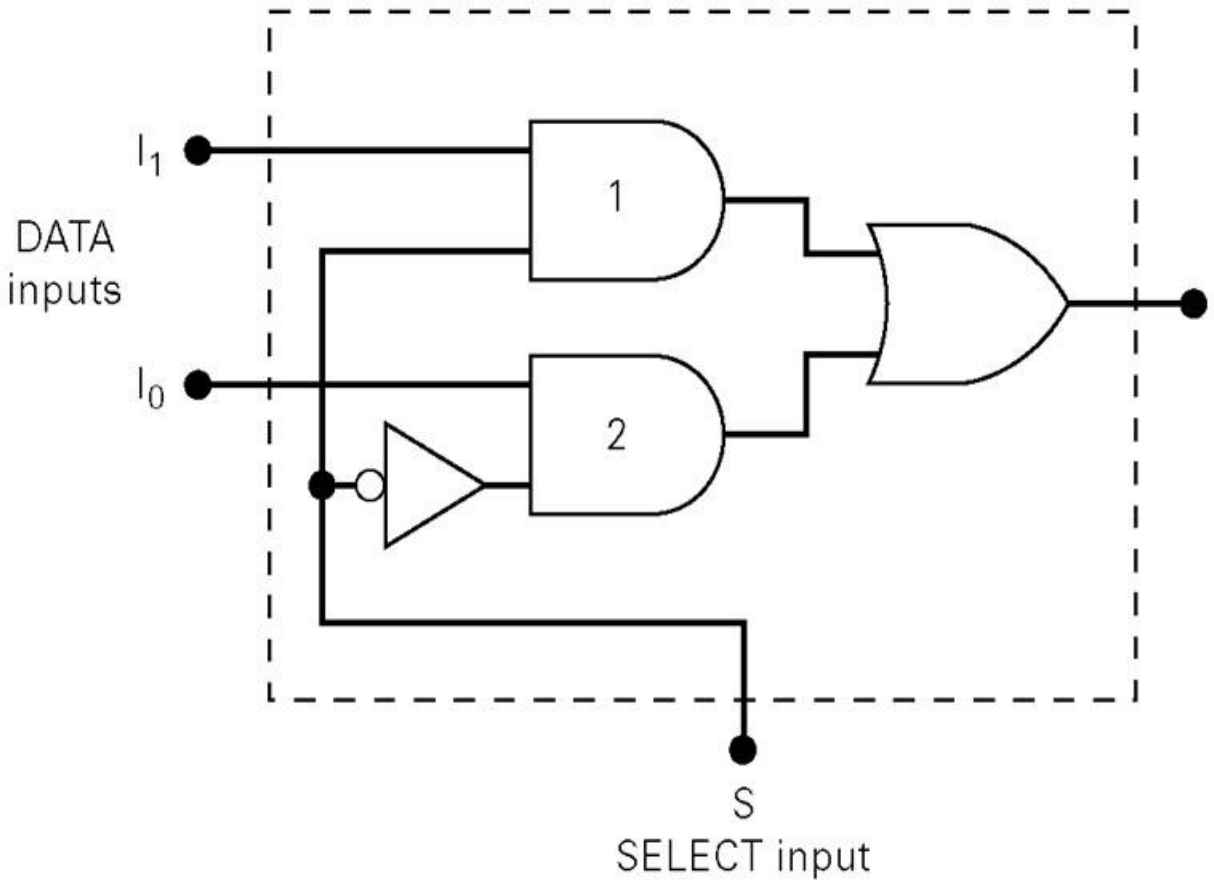


2:1 Multiplexer

Number of inputs=2
Number of outputs=1
Number of selection lines=1



2:1 Multiplexer

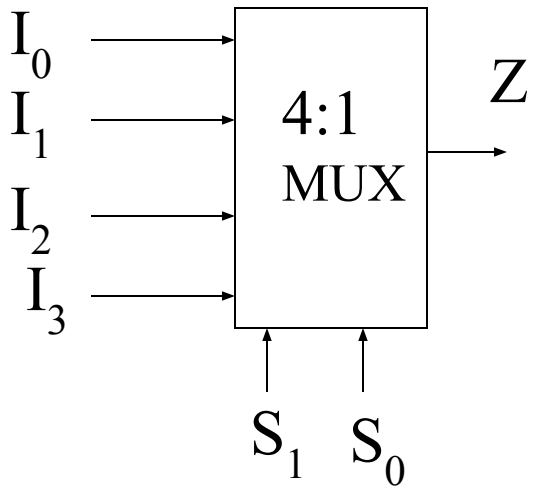


$$Z = I_0 \bullet \bar{S} + I_1 \bullet S$$

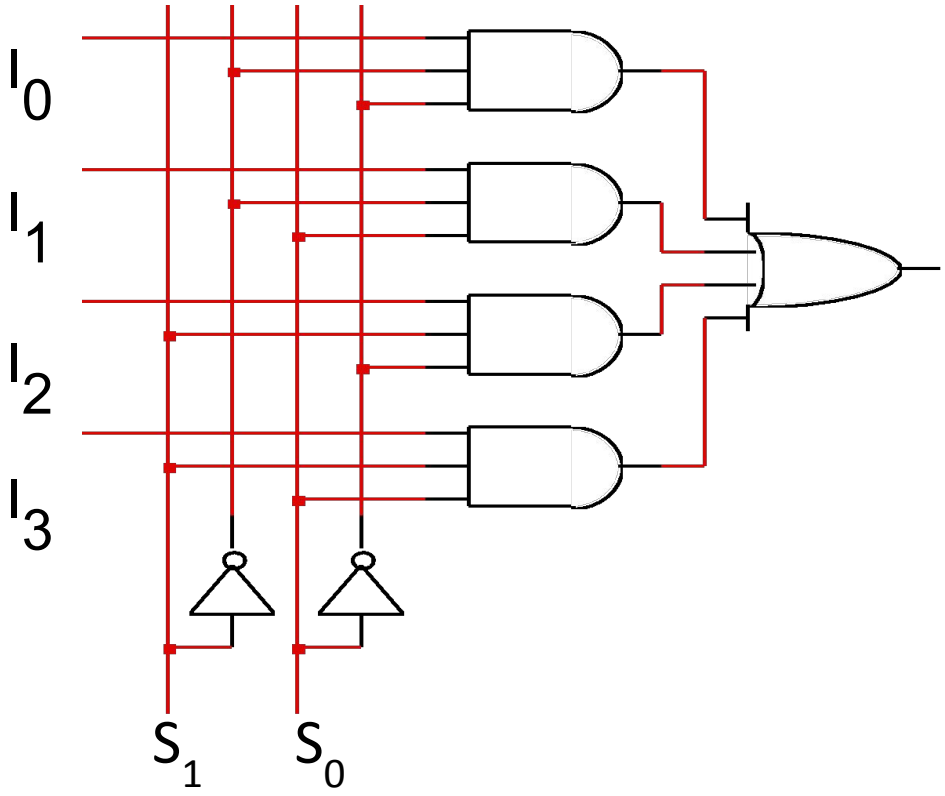
S	Output
0	$Z = I_0$
1	$Z = I_1$

4:1 Multiplexer

Number of inputs=4
Number of outputs=1
Number of selection lines=2



4:1 Multiplexer

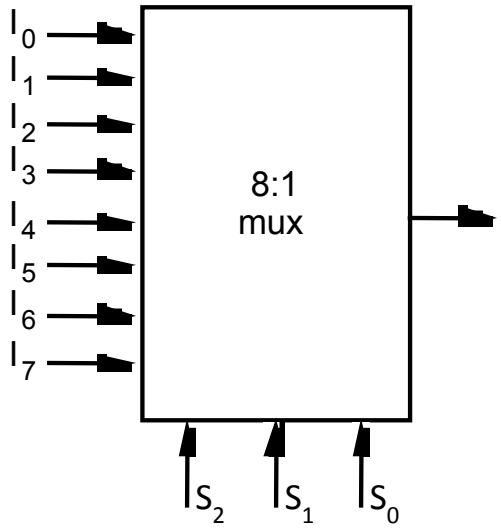


S_1	S_0	Output
0	0	$Z = I_0$
0	1	$Z = I_1$
1	0	$Z = I_2$
1	1	$Z = I_3$

$Z=S_0'S_1'I0+S_0S_1'I1+S_0'S_1I2+S_0S_1I3$

8:1 Multiplexer

Number of inputs=8
Number of outputs=1
Number of selection lines=3

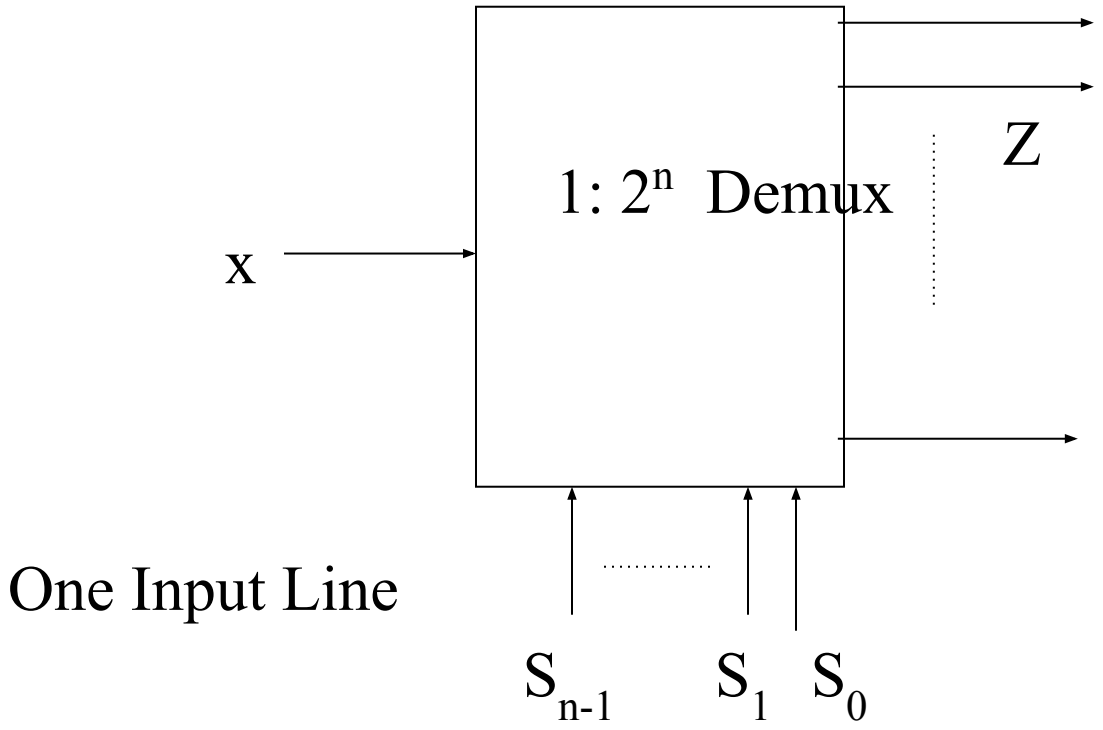


8:1 Multiplexer

S2	S1	S0	ENABLE E	OUTPUT Y
X	X	X	0	0
0	0	0	1	I_0
0	0	1	1	I_1
0	1	0	1	I_2
0	1	1	1	I_3
1	0	0	1	I_4
1	0	1	1	I_5
1	1	0	1	I_6
1	1	1	1	I_7

$$Y=S_2'S_1'S_0'I_0+S_2'S_1'S_0I_1+S_2'S_1S_0'I_2+S_2'S_1S_0I_3$$
$$+S_2S_1'S_0'I_4+S_2S_1'S_0\,I_5+S_2S_1S_0'\,I_6+S_2S_1S_0\,I_7$$

DEMULTIPLEXER



1:4 Demultiplexer

Input Line	S1	S0	Y_0	Y_1	Y_2	Y_3
I	0	0	I	0	0	0
I	0	1	0	I	0	0
I	1	0	0	0	I	0
I	1	1	0	0	0	I

1:8 Demultiplexer

Input Line	S2	S1	S0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
I	0	0	0	I	0	0	0	0	0	0	0
I	0	0	1	0	I	0	0	0	0	0	0
I	0	1	0	0	0	I	0	0	0	0	0
I	0	1	1	0	0	0	I	0	0	0	0
I	1	0	0	0	0	0	0	I	0	0	0
I	1	0	1	0	0	0	0	0	I	0	0
I	1	1	0	0	0	0	0	0	0	I	0
I	1	1	1	0	0	0	0	0	0	0	I

$$Y=S_2'S_1'S_0'I_0+S_2'S_1'S_0I_1+S_2'S_1S_0'I_2+S_2'S_1S_0I_3$$
$$+S_2S_1'S_0'I_4+S_2S_1'S_0\,I_5+S_2S_1S_0'\,I_6+S_2S_1S_0\,I_7$$

Decoders

- A decoder has
  - N inputs
  - 2^N outputs
- A decoder selects one of 2^N outputs by decoding the binary value on the N inputs.
  - Exactly one output will be active for each combination of the inputs.

Decoders

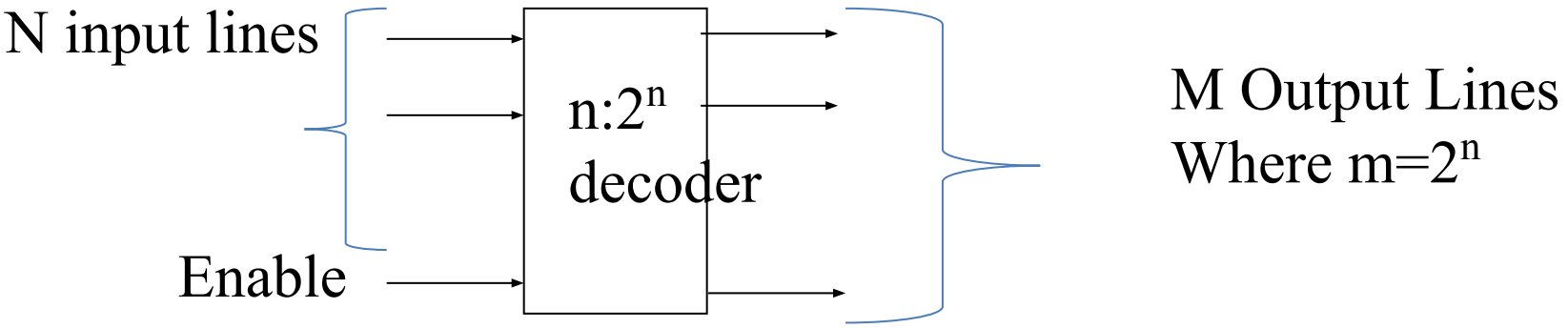
- A decoder has
  - N inputs
  - 2^N outputs
- Size of the Decoder-

“n X m lines”

“n to m lines”

Decoders

- A decoder is a multiple-input, multiple-output combinational logic circuit that converts coded inputs into coded outputs, where the input and output codes are different.

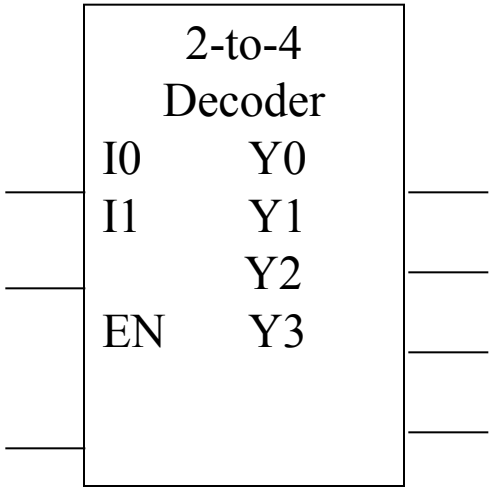


2 to 4 Binary Decoder

Truth table for a 2-to-4 binary decoder

Inputs			Outputs			
EN	I1	I0	Y3	Y2	Y1	Y0
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

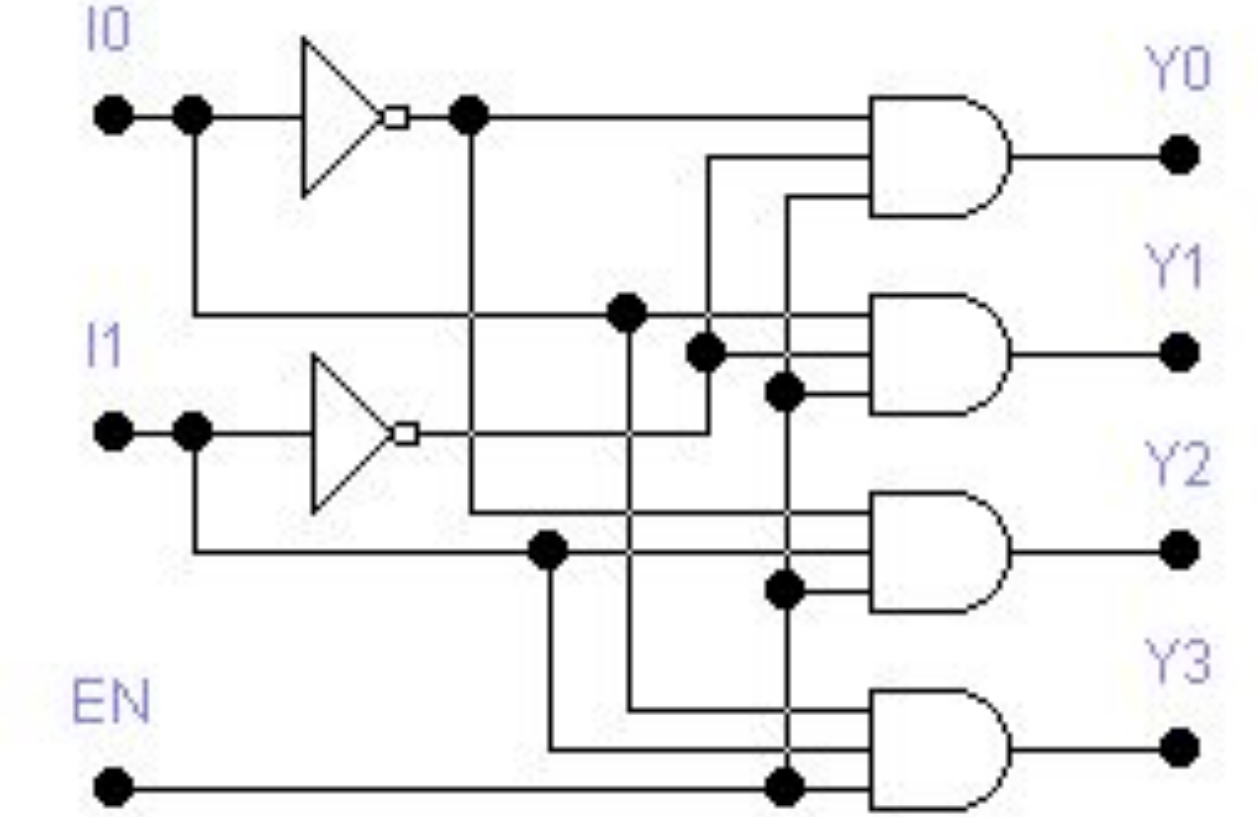
”don’t-care” notation



2 to 4 Decoder

$$Y0 = \overline{I1} \cdot \overline{I0} \cdot EN \qquad Y1 = \overline{I1} \cdot I0 \cdot EN$$

$$Y2 = I1 \cdot \overline{I0} \cdot EN \qquad Y3 = I1 \cdot I0 \cdot EN$$



3 TO 8 DECODER

No of Input Lines=3
No of Output Lines=8
Also 3x8 Decoder
3 line to 8 Line decoder
Binary to Octal Decoder

A	B	C	I7	I6	I5	I4	I3	I2	I1	I0
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

3 TO 8 DECODER

A	B	C	I7	I6	I5	I4	I3	I2	I1	I0
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

I0=A'B'C'
I1=A'B'C
I2=A'BC'
I3=A'BC
I4=AB'C'
I5=AB'C
I6=ABC'
I7=ABC

CODE CONVERTER

- Binary to Gray Code Converter
- Gray to Binary Code Converter
- BCD to Ex-3 Code Converter

FOUR BIT BINARY TO GRAY CODE CONVERTER –DESIGN (1)…

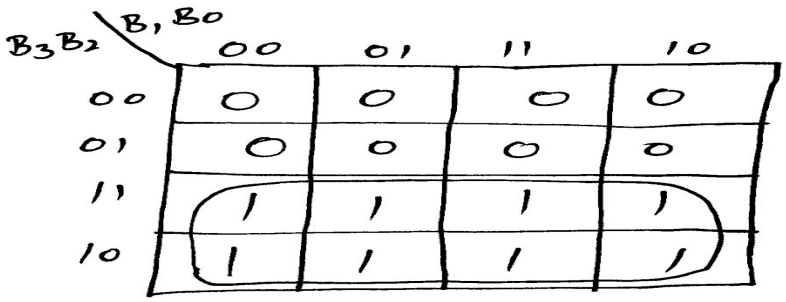
TRUTH TABLE:

Decimal	INPUT (BINARY)					OUTPUTS (GRAY CODE)			
	B3	B2	B1	B0		G3	G2	G1	G0
0	0	0	0	0		0	0	0	0
1	0	0	0	1		0	0	0	1
2	0	0	1	0		0	0	1	1
3	0	0	1	1		0	0	1	0
4	0	1	0	0		0	1	1	0
5	0	1	0	1		0	1	1	1
6	0	1	1	0		0	1	0	1
7	0	1	1	1		0	1	0	0
8	1	0	0	0		1	1	0	0
9	1	0	0	1		1	1	0	1
10	1	0	1	0		1	1	1	1
11	1	0	1	1		1	1	1	0
12	1	1	0	0		1	0	1	0
13	1	1	0	1		1	0	1	1
14	1	1	1	0		1	0	0	1
15	1	1	1	1		1	0	0	0

FOUR BIT BINARY TO GRAY CODE CONVERTER –DESIGN (2)…

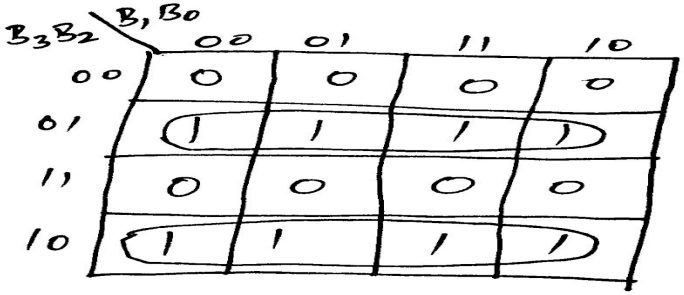
Simplification using K-maps:

G_3



$G_3 = B_3$

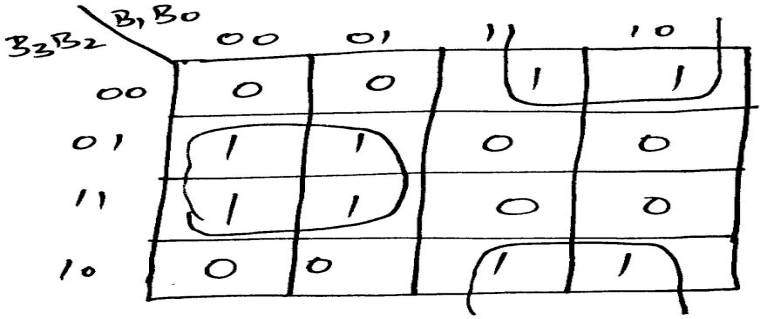
G_2



$G_2 = \overline{B_3} B_2 + B_3 \overline{B_2}$

$G_2 = B_3 \oplus B_2$

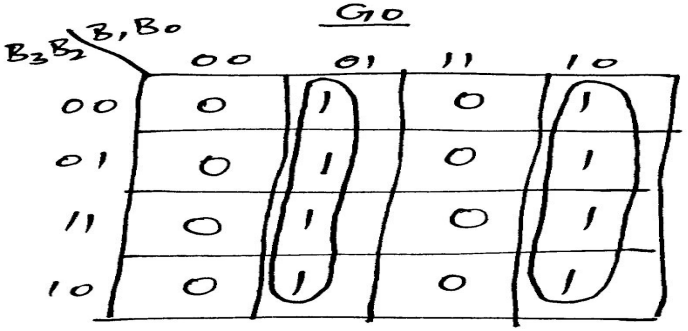
G_1



$G_1 = B_2 \overline{B_1} + \overline{B_2} B_1$

$G_1 = B_2 \oplus B_1$

G_0

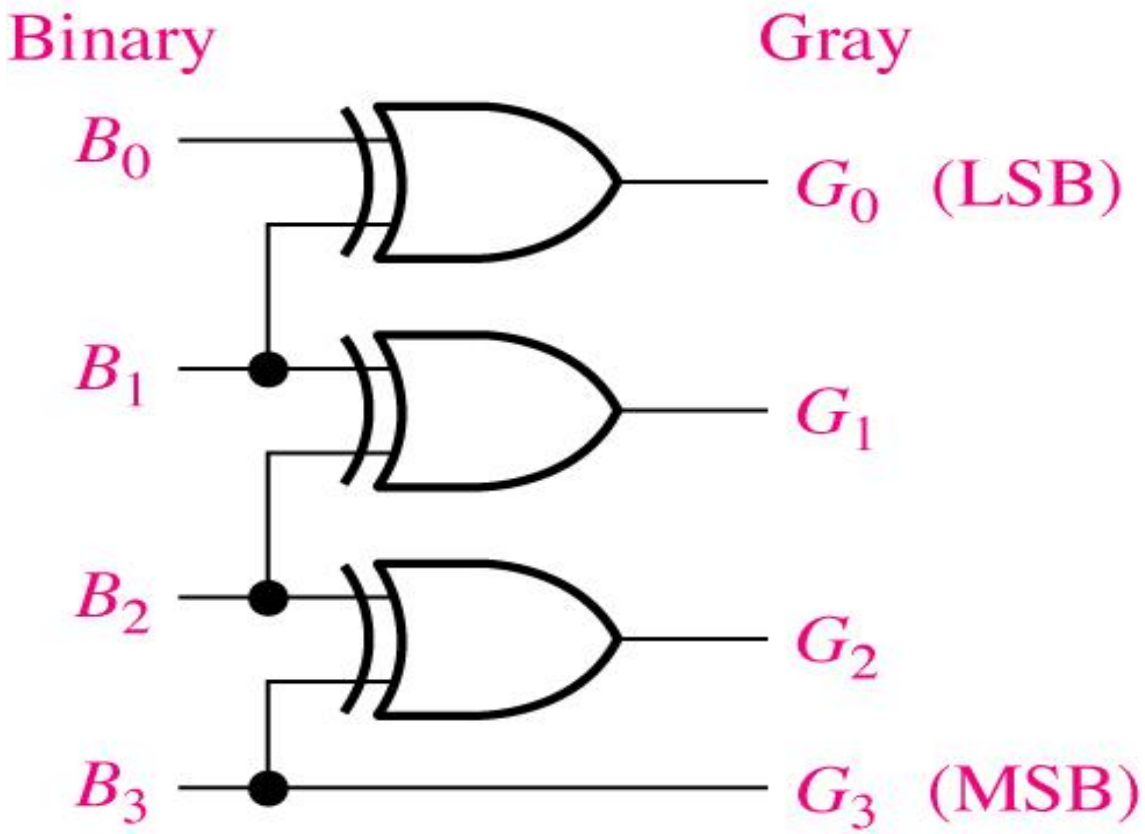


$G_0 = \overline{B_1} B_0 + B_1 \overline{B_0}$

$G_0 = B_1 \oplus B_0$

FOUR BIT BINARY TO GRAY CODE CONVERTER –DESIGN (3)

Logic Diagram:



FOUR BIT GRAY CODE TO BINARY CONVERTER –DESIGN (1)…

- **Truth Table:**

Decimal	INPUT (GRAY CODE)					OUTPUTS (BINARY)			
	G3	G2	G1	G0		B3	B2	B1	B0
0	0	0	0	0		0	0	0	0
1	0	0	0	1		0	0	0	1
2	0	0	1	1		0	0	1	0
3	0	0	1	0		0	0	1	1
4	0	1	1	0		0	1	0	0
5	0	1	1	1		0	1	0	1
6	0	1	0	1		0	1	1	0
7	0	1	0	0		0	1	1	1
8	1	1	0	0		1	0	0	0
9	1	1	0	1		1	0	0	1
10	1	1	1	1		1	0	1	0
11	1	1	1	0		1	0	1	1
12	1	0	1	0		1	1	0	0
13	1	0	1	1		1	1	0	1
14	1	0	0	1		1	1	1	0
15	1	0	0	0		1	1	1	1

FOUR BIT GRAY CODE TO BINARY CONVERTER –DESIGN (2)…

Simplification using K-Maps:

B_3

G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$$B_3 = G_3$$

B_2

G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$$B_2 = \overline{G_3}\,G_2 + G_3\overline{G_2}$$

$$B_2 = G_3 \oplus G_2$$

B_1

G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

$$B_1 = \overline{G_3}\,\overline{G_2}\,G_1 + G_3G_2G_1 + \overline{G_3}G_2\,\overline{G_1} + G_3\overline{G_2}\,\overline{G_1}$$

$$= G_1(\overline{G_3}\,\overline{G_2} + G_3G_2) + \overline{G_1}(\overline{G_3}\,G_2 + G_3\overline{G_2})$$

$$= G_1(\overline{G_3 \oplus G_2}) + \overline{G_1}(G_3 \oplus G_2)$$

$$= G_1 \oplus G_3 \oplus G_2$$

$$B_1 = G_1 \oplus B_2$$

FOUR BIT GRAY CODE TO BINARY CONVERTER –DESIGN (3)…

Simplification using K-Maps:

B_0

G_3G_2 \ G_1G_0	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

$$B_0 = \bar{G}_3\bar{G}_2\bar{G}_1G_0 + \bar{G}_3\bar{G}_2G_1\bar{G}_0 + \bar{G}_3G_2\bar{G}_1\bar{G}_0 + \bar{G}_3G_2G_1G_0 + G_3G_2\bar{G}_1G_0 + G_3G_2G_1\bar{G}_0 + G_3\bar{G}_2\bar{G}_1\bar{G}_0 + G_3\bar{G}_2G_1G_0$$

$$= \bar{G}_3\bar{G}_2(\bar{G}_1G_0 + G_1\bar{G}_0) + \bar{G}_3G_2(\bar{G}_1\bar{G}_0 + G_1G_0) + G_3G_2(\bar{G}_1G_0 + G_1\bar{G}_0) + G_3\bar{G}_2(\bar{G}_1\bar{G}_0 + G_1G_0)$$

$$= \bar{G}_3\bar{G}_2(G_1 \oplus G_0) + \bar{G}_3G_2(\overline{G_1 \oplus G_0}) + G_3G_2(G_1 \oplus G_0) + G_3\bar{G}_2(\overline{G_1 \oplus G_0})$$

$$= (G_1 \oplus G_0)(\bar{G}_3\bar{G}_2 + G_3G_2) + \overline{(G_1 \oplus G_0)}(\bar{G}_3G_2 + G_3\bar{G}_2)$$

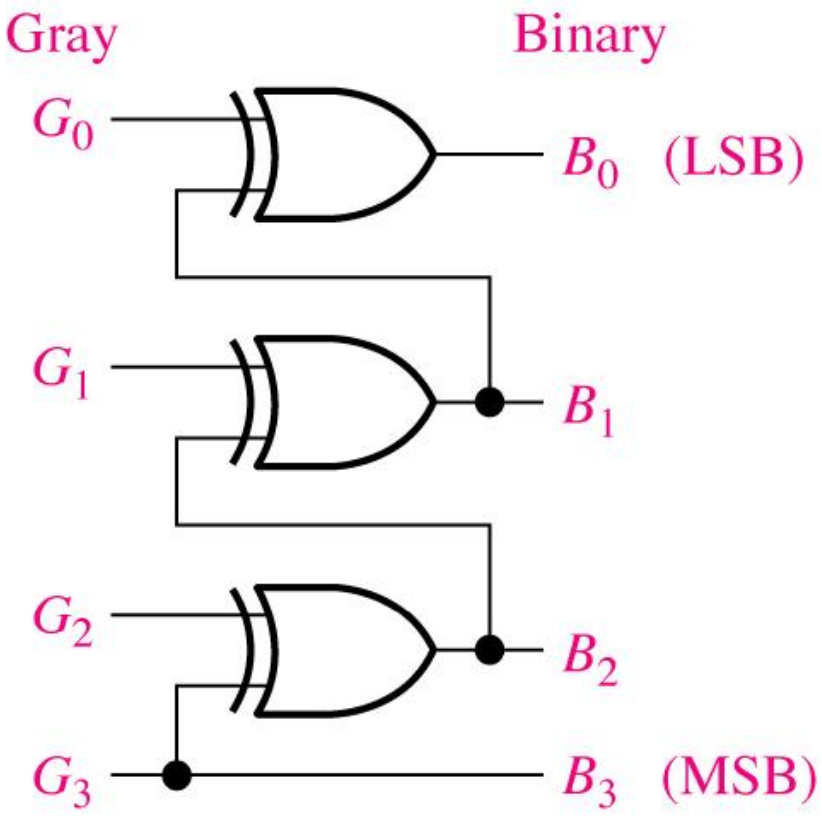
$$= (G_1 \oplus G_0)(\overline{G_3 \oplus G_2}) + \overline{(G_1 \oplus G_0)}(G_3 \oplus G_2)$$

$$= G_0 \oplus G_1 \oplus G_2 \oplus G_3$$

$$\boxed{B_0 = G_0 \oplus B_1}$$

FOUR BIT GRAY CODE TO BINARY CONVERTER –DESIGN (4)

Logic Diagram:



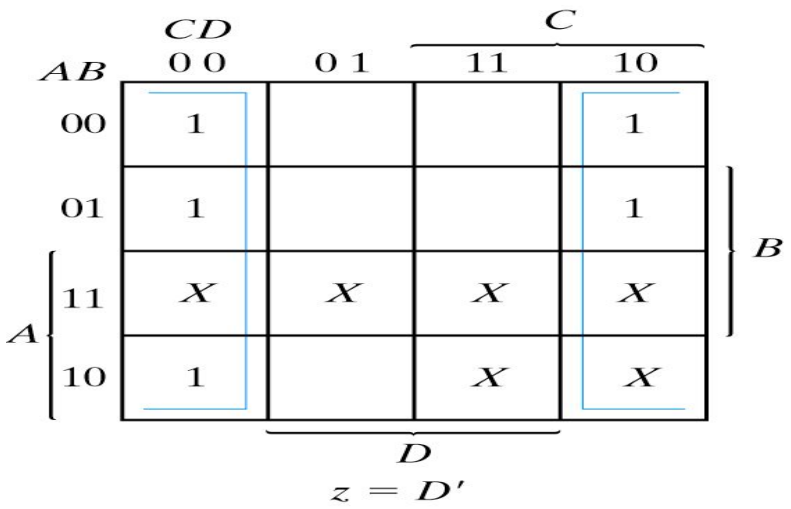
BCD to XS 3 code converter- Design (1)...

TRUTH TABLE FOR BCD TO XS3 CODE CONVERTER:

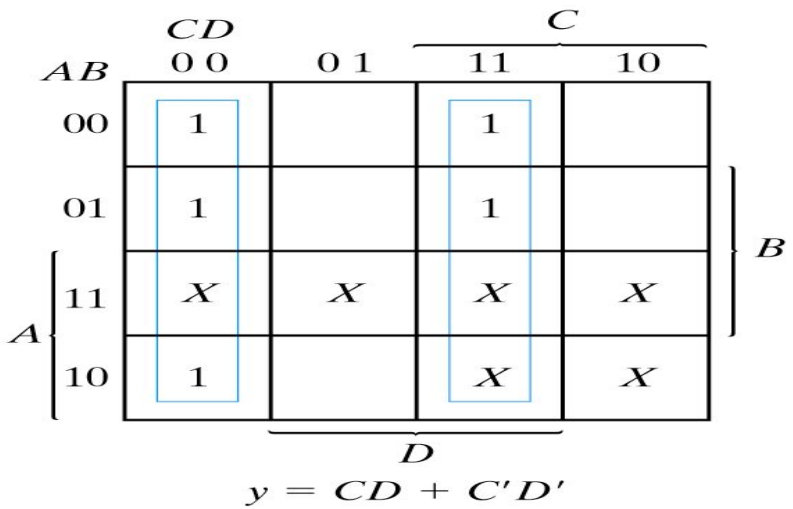
Input (Std BCD code)					Output (XS3 Code)			
A	**B**	**C**	**D**		w	x	y	z
0	**0**	**0**	**0**		**0**	**0**	**1**	**1**
0	**0**	**0**	**1**		**0**	**1**	**0**	**0**
0	**0**	**1**	**0**		**0**	**1**	**0**	**1**
0	**0**	**1**	**1**		**0**	**1**	**1**	**0**
0	**1**	**0**	**0**		**0**	**1**	**1**	**1**
0	**1**	**0**	**1**		**1**	**0**	**0**	**0**
0	**1**	**1**	**0**		**1**	**0**	**0**	**1**
0	**1**	**1**	**1**		**1**	**0**	**1**	**0**
1	**0**	**0**	**0**		**1**	**0**	**1**	**1**
1	**0**	**0**	**1**		**1**	**1**	**0**	**0**
1	**0**	**1**	**0**		**X**	**X**	**X**	**X**
1	**0**	**1**	**1**		**X**	**X**	**X**	**X**
1	**1**	**0**	**1**		**X**	**X**	**X**	**X**
1	**1**	**1**	**0**		**X**	**X**	**X**	**X**
1	**1**	**1**	**1**		**X**	**X**	**X**	**X**

BCD to XS 3 code converter- Design (2)...

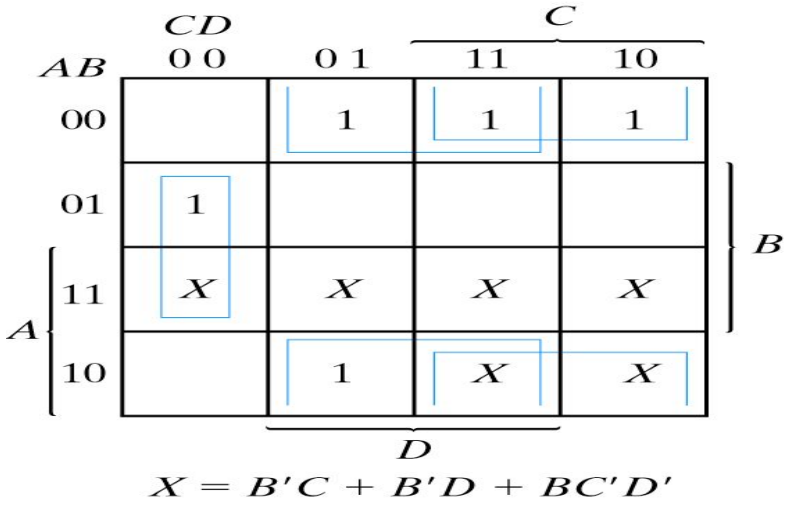
K-maps for simplification and simplified Boolean expressions



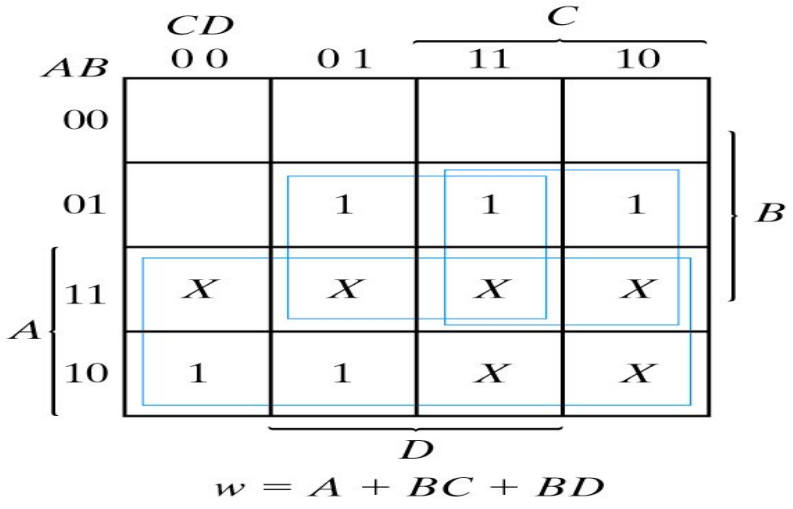
$z = D'$



$y = CD + C'D'$



$X = B'C + B'D + BC'D'$



$w = A + BC + BD$

BCD to XS 3 code converter- Design (3)...

- After the manipulation of the Boolean expressions for using common gates for two or more outputs, logic expressions can be given by

  z=D’

  y=CD+C’D’ = CD + (C+D)’

  x= B’C + B’D + BC’D’ = B’(C+D) + BC’D’

  w= A + BC + BD = A + B (C+D)

BCD to XS 3 code converter- Design (4)