Digital Techniques Lecture Notes

Lecturer

Ibtesam Raheem Karhiy Al-Saedi

Asst. Prof. Dr. Eng.

Communication Engineering Deptrament

University of Technology (UOT)
Baghdad-Iraq

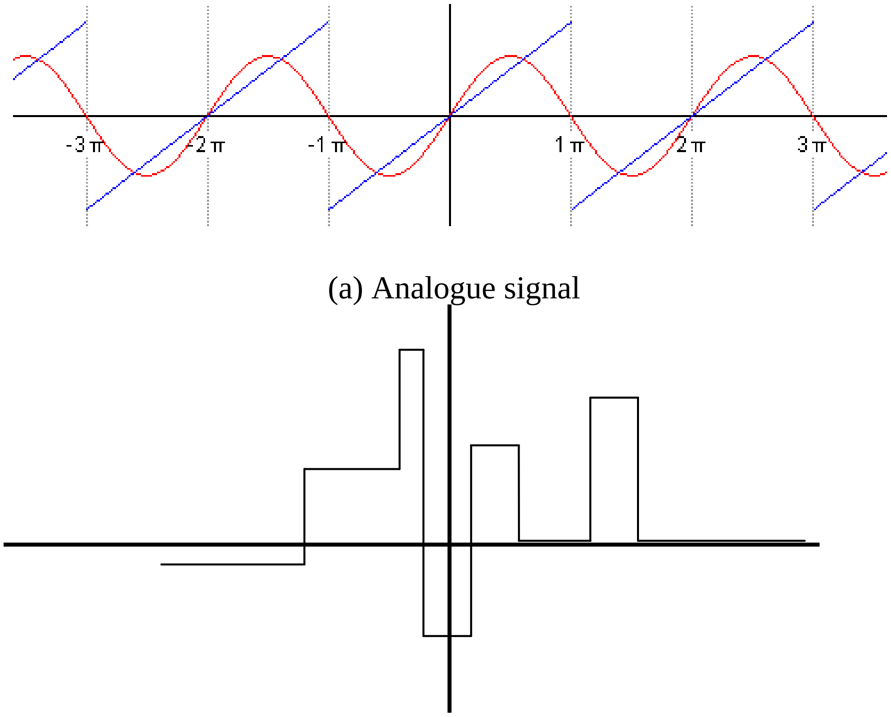
Lecture 1 : Digital Signal and Binary Numbers

To make some sense of the 'analogue vs. digital' debate, let's firstly establish what the two terms mean. In the case of digital electronics, we're talking about two different methods of sending an electronic signal from A to B.

ANALOGUE signals are continuous, and can take any value.

DIGITAL signals encode values into binary numbers. As a binary number is made up entirely from 0's and 1's, it may be transmitted in the form of electronic *on/off* pulses (on =1, off =0). When these pulses are received, they are processed. A digital signal is made up of discretely variable physical quantities.

Figure 1 explains both Analogue and Digital signals



(a) Analogue signal

(B)Discrete Signal

Figure 1 (A) Analoge signal , (B) Discrete Signal

BITS and BYTES, kilobytes, megabytes and gigabytes

Any discussion of computer technology will usually use some - or all - of the following terms. They all have relatively straightforward definitions, however, and refer to varying quantities of computer memory:

a BIT (a binary digit) = one *on/off* space in memory - recognised as a 0 or 1.

a BYTE = eight bits, and can therefore hold any decimal value from 0 (00000000) to 255 (11111111).

a KILOBYTE (K Byte) = about one thousand bytes. In fact, 1KB = 1024 bytes = 2^{10} bytes, similarly...

a MEGABYTE (M Byte) = 1024 kilobytes, and finally

a GIGABYTE (G Byte) = 1024 megabytes.

BINARY numbers

The following table gives the binary equivalent values for 0-15 (decimal):

DECIMAL	BINARY	DECIMAL	BINARY
10^2 10^2 10^2	2^3 2^2 2^1 2^0	10^2 10^2 10^2	2^3 2^2 2^1 2^0
0 0 0	0 0 0 0	0 0 8	1 0 0 0
0 0 1	0 0 0 1	0 0 9	1 0 0 1
0 0 2	0 0 1 0	0 1 0	1 0 1 0
0 0 3	0 0 1 1	0 1 1	1 0 1 1
0 0 4	0 1 0 0	0 1 2	1 1 0 0
0 0 5	0 1 0 1	0 1 3	1 1 0 1
0 0 6	0 1 1 0	0 1 4	1 1 1 0
0 0 7	0 1 1 1	0 1 5	1 1 1 1

BINARY conversion and 'arithmetic'

To manually convert a decimal (base 10) number to a binary (base 2) number, successive division of the decimal number by 2 must be performed:

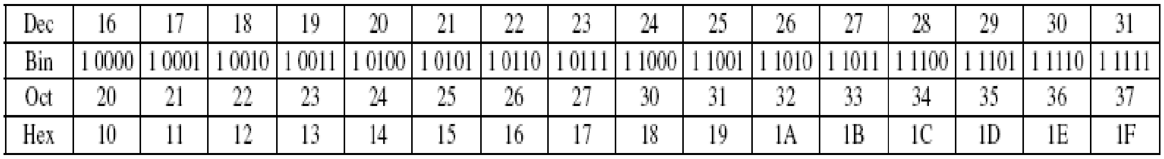
```
  600 / 2 = 300 rem. 0          <-- LSB (least
significant bit)
  300 / 2 = 150 rem. 0\
  150 / 2 =  75 rem. 0\\
   75 / 2 =  37 rem. 1\\\
   37 / 2 =  18 rem. 1\\\\
   18 / 2 =   9 rem. 0\\\\\
    9 / 2 =   4 rem. 1\\\\\\
    4 / 2 =   2 rem. 0\\\\\\\
    2 / 2 =   1 rem. 0\\\\\\\\
    1 / 2 =   0 rem. 1\\\\\\\\\ <-- MSB (most significant
bit)
                      \\\\\\\\\\
                       1001011000  <----- all of the
remainders from the division are then arranged in  reverse order, from
MSB to LSB to form the correct binary sequence.
```

Some of Cods Numbers

Decimal No.	Binary No.	8421 BCD	Octal No.	Hexadecimal No.
10^3 10^2 $10^1 10^0$	2^3 2^2 $2^1 2^0$		8^3 8^2 $8^1 8^0$	16^3 16^2 $1^1 16^0$
0	000000	0000 0000	00	00
1	000001	0000 0001	01	01
2	000010	0000 0010	02	02
3	000011	0000 0011	03	03
4	000100	0000 0100	04	04
5	000101	0000 0101	05	05
6	000110	0000 0110	06	06
7	000111	0000 0111	07	07
8	001000	0000 1000	10	08
9	001001	0000 1001	11	09
10	001010	0001 0000	12	0A
11	001011	0001 0001	13	0B
12	001100	0001 0010	14	0C
13	001101	0001 0011	15	0D
14	001110	0001 0100	16	0E
15	001111	0001 0101	17	0F
16	010000	0001 0110	20	10
17	010001	0001 0111	21	11
18	010010	0001 1000	22	12
19	010011	0001 1001	23	13
20	010100	0010 0000	24	14

Examples:

Decimal, Binary, Octal and Hexadecimal Numbers from $(16)_{10}$ to $(31)_{10}$

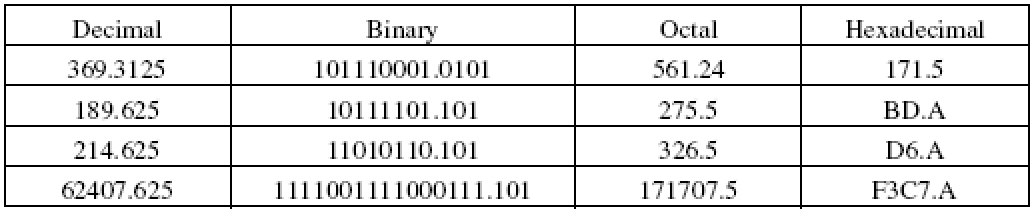


Dec	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Bin	1 0000	1 0001	1 0010	1 0011	1 0100	1 0101	1 0110	1 0111	1 1000	1 1001	1 1010	1 1011	1 1100	1 1101	1 1110	1 1111
Oct	20	21	22	23	24	25	26	27	30	31	32	33	34	35	36	37
Hex	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F

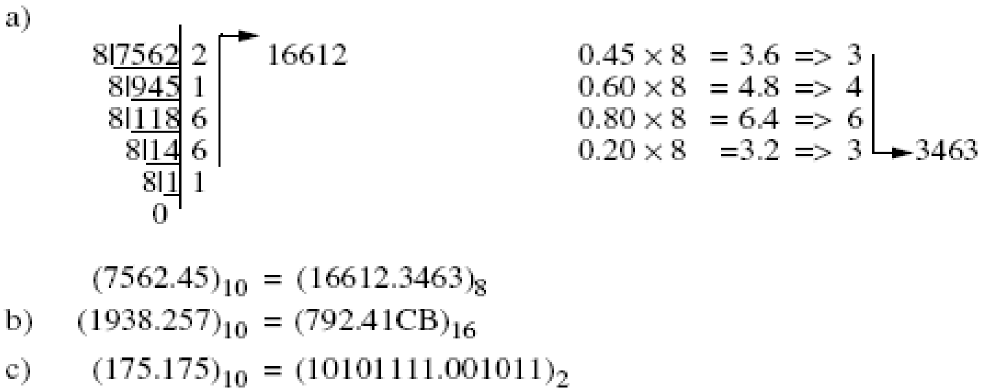
$$(1001101)_2 = 2^6 + 2^3 + 2^2 + 2^0 = 77$$

$$(1010011.101)_2 = 2^6 + 2^4 + 2^1 + 2^0 + 2^{-1} + 2^{-3} = 83.625$$

$$(10101110.1001)_2 = 2^7 + 2^5 + 2^3 + 2^2 + 2^1 + 2^{-1} + 2^{-4} = 174.5625$$



Decimal	Binary	Octal	Hexadecimal
369.3125	101110001.0101	561.24	171.5
189.625	10111101.101	275.5	BD.A
214.625	11010110.101	326.5	D6.A
62407.625	1111001111000111.101	171707.5	F3C7.A



a)

$$\begin{array}{rl} 8\,|\underline{7562} & 2 \\ 8\,|\underline{945} & 1 \\ 8\,|\underline{118} & 6 \\ 8\,|\underline{14} & 6 \\ 8\,|\underline{1} & 1 \\ 0 & \end{array} \rightarrow 16612$$

$$\begin{array}{l} 0.45 \times 8 = 3.6 \Rightarrow 3 \\ 0.60 \times 8 = 4.8 \Rightarrow 4 \\ 0.80 \times 8 = 6.4 \Rightarrow 6 \\ 0.20 \times 8 = 3.2 \Rightarrow 3 \end{array} \rightarrow 3463$$

$(7562.45)_{10} = (16612.3463)_8$

b) $(1938.257)_{10} = (792.41CB)_{16}$

c) $(175.175)_{10} = (10101111.001011)_2$

Some of BCD Cods

DECIMAL	2421	2421	5421	-2841	5043210	EXCESS-3	SEVEN - SEGEMENT
0	0000	0000	0000	0000	0100001	0011	1111110
1	0001	0001	0001	0001	0100010	0100	0110000
2	0010	0010	0010	1010	0100100	0101	1101101
3	0011	0011	0011	1011	0101000	0110	1111001
4	0100	01000	0100	1010	0110000	0111	0110011
5	0101	1011	1000	0011	1000001	1000	0011011
6	0110	1100	1001	1100	1000010	1001	0011111
7	0111	1191	1010	1101	1000100	1010	1110000
8	1110	1110	1011	0100	1001000	1011	1111111
9	1111	1111	1100	0101	1010000	1100	1110011

Logic Gates and Boolen Algebra:

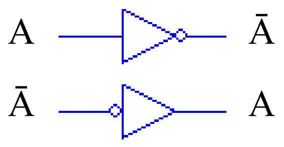
Logic Gates: In digital electronics a gate is "a circuit with one output and two or more inputs". An output of the gate occurs only for certain combination of the input signal.

1. Inverter or NOT Gate:

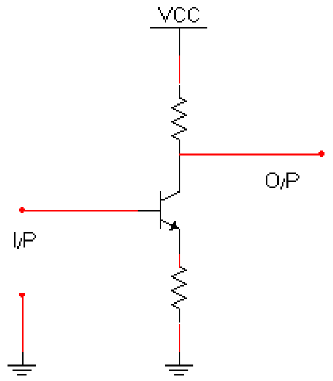
Truth Table

I/P	O/P
A	Ā
0	1
1	0

The Sample



NOT Circuit (Inverter)

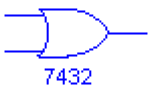


2. OR Gate:

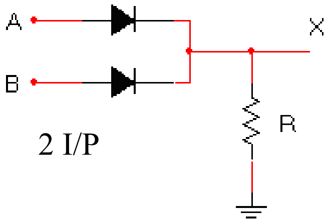
Truth Table

I/P		O/P
B	A	X=A+B
0	0	0
0	1	1
1	0	1
1	1	1

The Sample



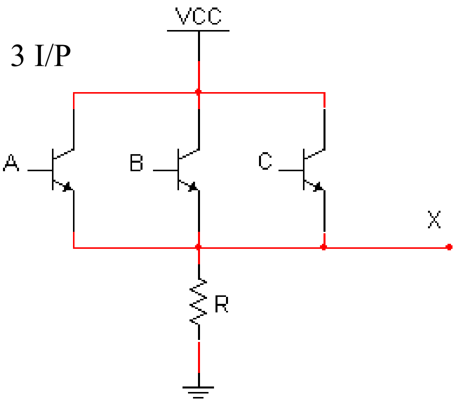
Diode-Resister Logic Circuit



Truth Table

I/P			O/P
C	B	A	X=A+B+C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

TTL Logic circuit

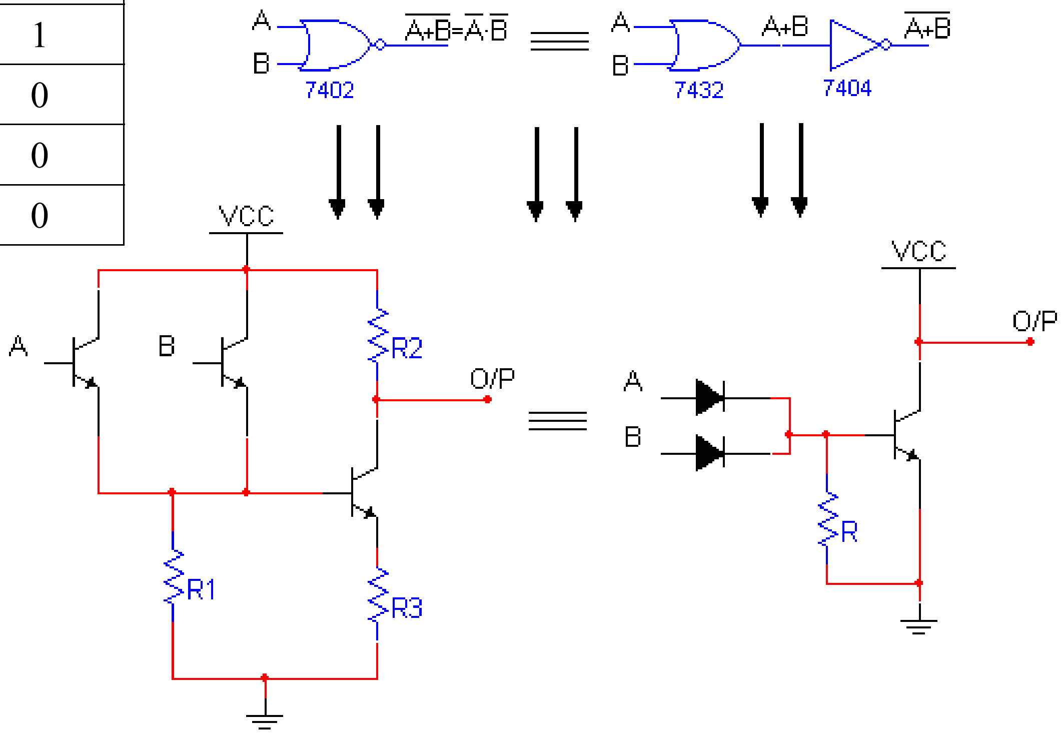


3. NOR Gate:

Truth Table

I/P		O/P
B	A	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

The Sample

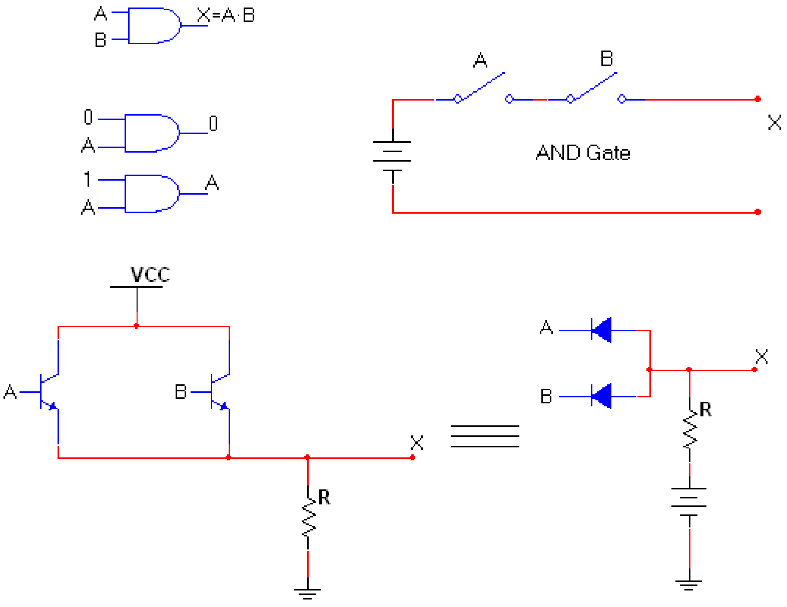


4. AND Gate:

Truth Table

I/P		O/P
B	A	X=A.B
0	0	0
0	1	0
1	0	0
1	1	1

The Sample

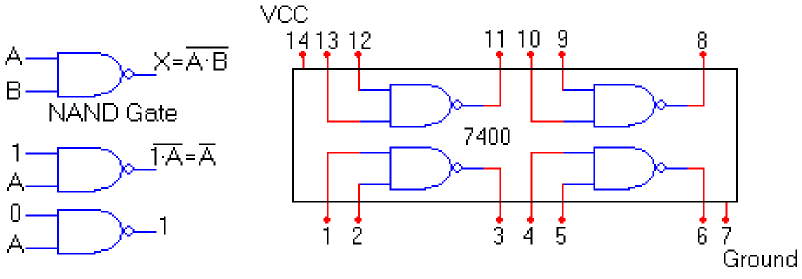


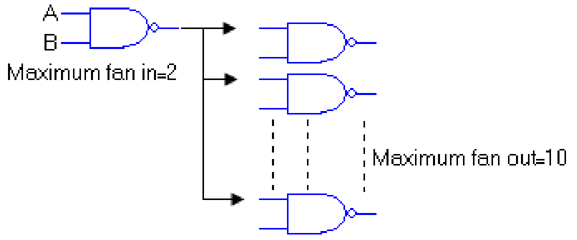
5. NAND Gate:

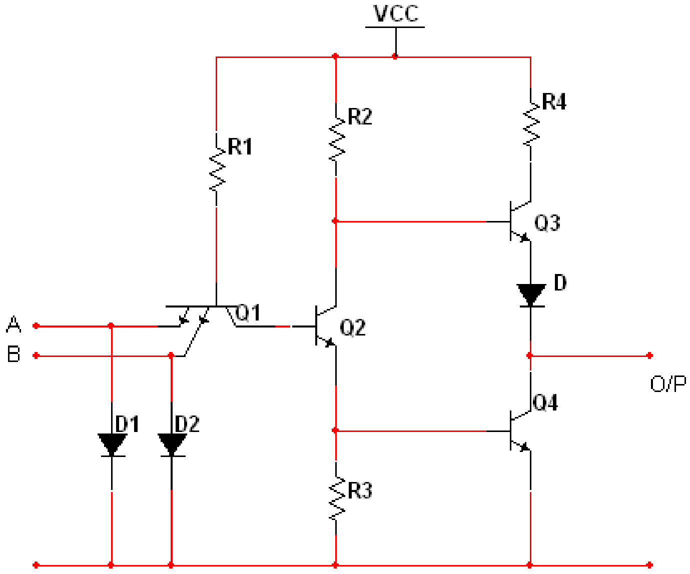
Truth Table

I/P		O/P
B	A	$X=\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

The Sample







7400 NAND gate with 2 input
7410 NAND gate with 3 input
7420 NAND gate with 4 input
7430 NAND gate with 8 input

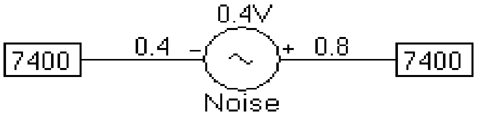
Standard TTL circuit

الغرض من الربط الطوطي الحصول على بوابة إخراج واطئة و وجود Q3 لتسريع من حالة ON إلى OFF
وتعتبر هذه الدائرة من الدوائر القياسية ومواصفاتها ك TTLG هي:

1. تعمل بدرجة حرارة (0 إلى 70).

2. V1L=0-8 V
V1H=2 V
V0L=0-4 V
V0H=2-4 V

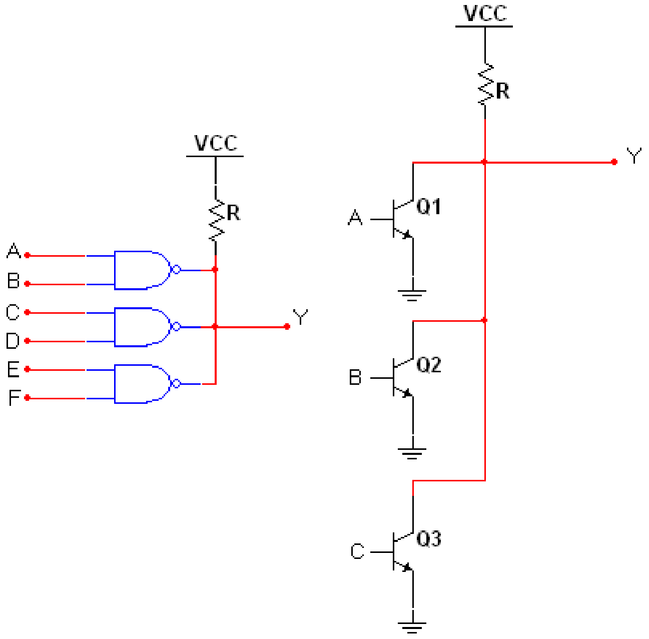
3. Fan-Out =10
Fan-In = 10



4. Noise Immunity = 0.4

5. عند رفع Q3تتحول الدائرة الى ما يسمى NAND gate ذات المجمع المفتوح open collector ولعدم وجود TrQ3 يجب ربط مقاومة تصعيد خارجية من O/P و VCC حتى تعمل الدائرة وتتراوح قيمة المقاومة من بعض مئات الى الاف الأومات.

6. ابطئ من الشكل الطوطي ولكن السبب في الحاجة الى استخدامها هو امكانية الربط المباشر لاطراف الاخراج بالاشتراك مع مقاومة التصعيد الخارجية كما في الشكل التالي:



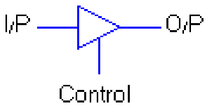
7. يمكن ان تكون الدائرة السابقة عدة انواع:

1. Standard TTL
2. Low Power TTL
3. High speed TTL
4. Schottky-clamped TTL
5. 3-Stable O/P

6. State logic gate:

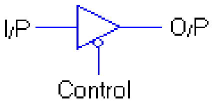
   1. Low control
   2. High control

   with inverter
   without inverter



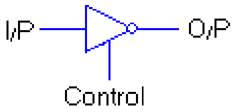
3-State(Tri-State)
Active High without inverter

C	I/P	O/P
0	0	H.Z
0	1	H.Z
1	0	0
1	1	1



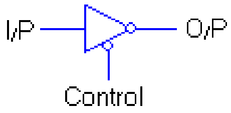
3-State(Tri-State)
Active Low without inverter

C	I/P	O/P
0	0	0
0	1	1
1	0	H.Z
1	1	H.Z



3-State(Tri-State)
Active High with inverter

C	I/P	O/P
0	0	H.Z
0	1	H.Z
1	0	1
1	1	0



3-State(Tri-State)
Active Low with inverter

C	I/P	O/P
0	0	1
0	1	0
1	0	H.Z
1	1	H.Z

مزايا البوابات السابقة:

1. استهلاك قدرة عالية.
2. سرعة العمل.
3. توفرها.

هنالك نوع أخر من الدوائر يدعى (CMOS(Complementary Logic Metal Oxide Semiconductor وتمتاز بالأتي:

1. تستخدم ترانزستور FET
2. استهلاك قدرة اقل من TTL
3. ابسط في التركيب
4. الدائرة المتكاملة يكون حجمها اصغر من TTL ولكنها أبطئ

أنواع أخرى من الدوائر الرقمية:

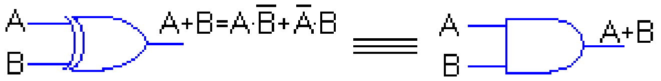
1. I^2L
تستخدم هذه الدوائر و الترانزستورات ثنائية القطبية Bipolar ولها استخدامات في الدوائر المتكاملة الكبيرة LSI Large Scale Integration
2. CCD
Charge Coupled Devices مشابهة لدوائر CMOS

7. EX-OR gate Exclusive OR gate:

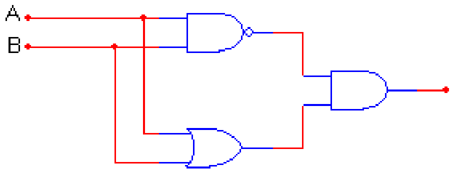
Truth Table

I/P		O/P
B	A	X=A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

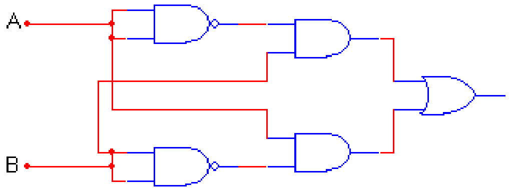
The Sample



Another way to implement E-XOR:



$$(\overline{AB})(\overline{A}+\overline{B})= (A+B)\ (A+B)$$
$$=A\overline{A}+\overline{A}B+\overline{B}A+\overline{B}B=\overline{A}B+\overline{B}A=A\oplus B$$



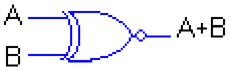
$$\overline{A}B+\overline{B}A=A\oplus B$$

8. Exclusive-NOR-gate(EX-NOR):

Truth Table

I/P		O/P
B	A	$X=\overline{A\oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

The Sample

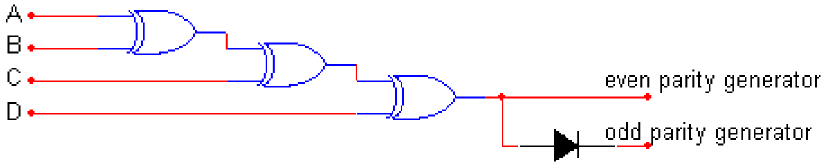


$=\overline{A.\bar{B}+\bar{A}.B}=\overline{A\oplus B}$

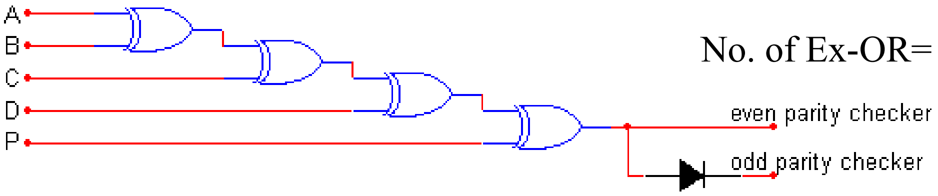
H.W: implement OR gate and NAND gate and inverter using NOR gate only?

SOME applications of EX-OR gate

1. Parity generator and parity checker to determine the parity of a binary number (i.e. odd or even) an EX-OR may be used to implement parity follows:



4-bit parity generator



No. of Ex-OR= (No. of bit)-1

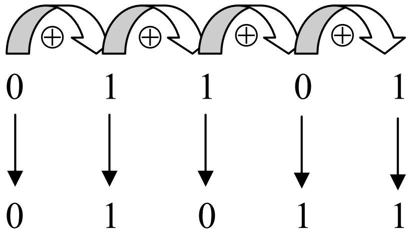
4-bit parity checker

H.W: Find the parity of the following binary number and implement the parity checker, 11011101, 110111, and 1101110110?

C	B	A	P even	P odd
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0

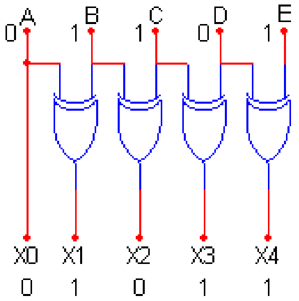
2. Binary to GRAY code conversion:
   The main feature of gray code is that each gray number differs from the preceding gray number by a single bit

   TO generate the GRAY code:
   1. The first gray digit is the same as binary digit.
   2. Add each pair of adjacent–bits to get the next digit using modulo-2 (EX-OR) addition:



No. of Ex-OR= (No. of bit)-1

Binary
Gray



H.W: Using binary to gray converter find the gray of the following binary number 1011011, 111011000, 11011011101?

H.W: Implement EX-OR by using:
1. NAND gate only
2. NOR gate only

Boolean algebra and demorgan's theorem

1. Basic laws:
   a) $X\cdot 0=0$
   b) $X+0=X$
   c) $X\cdot 1=X$
   d) $X+1=1$
   e) $X\cdot X=X$
   f) $X+X=X$
   g) $X\cdot\overline{X}=0$
   h) $X+\overline{X}=1$
   i) $\overline{\overline{X}}=X$
2. Commutative law:
   a) $X\cdot Y=Y\cdot X$
   b) $X+Y=Y+X$
3. Associative low:
   a) $(X+Y)+Z=X+(Y+Z)$
   b) $X\cdot Y\cdot Z=X\cdot(Y\cdot Z)$

4. Demorgan's theorem:
   a) $\overline{X\cdot Y}=\overline{X}+\overline{Y}$
   b) $\overline{X+Y}=\overline{X}\cdot\overline{Y}$

Proofing:

$X\cdot Y$	$\overline{X\cdot Y}$	$\overline{X}+\overline{Y}$	$\overline{X+Y}$	$\overline{X}\cdot\overline{Y}$
0 0	$\overline{0\cdot 0}=1$	$1+1=1$	$\overline{0+0}=1$	$1\cdot 1=1$
0 1	$\overline{0\cdot 1}=1$	$1+0=1$	$\overline{0+1}=0$	$1\cdot 0=0$
1 0	$\overline{1\cdot 0}=1$	$0+1=1$	$\overline{1+0}=0$	$0\cdot 1=0$
1 1	$\overline{1\cdot 1}=0$	$0+0=0$	$\overline{1+1}=0$	$0\cdot 0=0$

Proof by truth table of Demorgan's relationship

5. Distributive law:
   a) $X\cdot(Y+Z)=(X\cdot Y)+(X.Z)$
   b) $X+(Y\cdot Z)=(X+Y)(X+Z)$

Proofing:

$X+(Y\cdot Z)=\overline{\overline{X}\cdot\overline{(Y\cdot Z)}}$ Demorgan's theorem

$=\overline{\overline{X}\cdot(\overline{Y}+\overline{Z})}=\overline{(\overline{X}\cdot\overline{Y})}\cdot\overline{(\overline{X}\cdot\overline{Z})}$ Distributive law

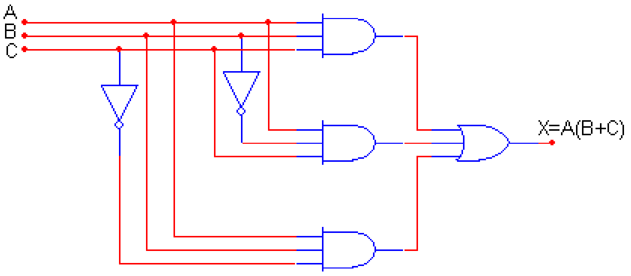
$=(X+Y)\cdot(X+Z)$ Distributive law

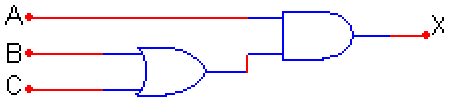
6. Miscellaneous theorems
   a) $A+AB=A$
   b) $A(A+B)=A+AB=A(1+B)=A$
   c) $(A+B)(A+C)=A+BC$
   d) $A+\overline{A}B=A+B$
   e) $A(\overline{A}+B)=AB$
   f) $(A+B)(\overline{A}+C)=AC+\overline{A}B$
   g) $AB+\overline{A}C=(A+C)(\overline{A}+B)$

Example: Show that $X=ABC+\overline{A}BC+AB\overline{C}$ can be simplified to $X=A(B+C)$ and implement the logic circuit for each expression:

$X=AC(B+\overline{B})+AB\overline{C}$ $\quad B+\overline{B}=1$

$=AC+AB\overline{C}=A(C+B\overline{C})=A(B+C)$





Simplified Circuit

H.W:

1. Simplified $X=(A+B)(A+\overline{B})(\overline{A}+B)$ and implement the logic circuit?
2. Show how NAND gates are used to implement $X=AB+CD$ (Use NAND gate with 2 input only)?

H.W:

1. Give the O/P of logic circuits:

$$X=CA+BC(A+B)+B$$

   a) Implement the logic gate circuit?
   b) Simplify the expression and implement the logic circuit?
   c) What is the value of X when A=0, B=1, C=0
      A=0, B=0, C=1

2. $\overline{A}\cdot\overline{C}+A\cdot\overline{B}\cdot\overline{C}=\overline{A}\cdot\overline{C}+\overline{B}\cdot\overline{C}$
3. $F=(A.\overline{C})+(A.B.C)+(A.C.\overline{D})+(C.D)$
4. $F=\overline{\overline{X}\cdot(Y+\overline{Z})}\cdot(X+\overline{Y}+Z)\cdot\overline{(\overline{X}\cdot\overline{Y}\cdot\overline{Z})}$
5. $F=\overline{A}\cdot\overline{C}\cdot(B+B\cdot D)+(A\cdot\overline{C}\cdot D)$

Solve of (4):

$F=X+\overline{(Y+\overline{Z})}\cdot(X+\overline{Y}+Z)\cdot(X+Y+Z)$ Demorgan's theorem

$=\{X+(\overline{Y}\cdot Z)\}\cdot\{X+Z+(Y\cdot\overline{Y})\}$

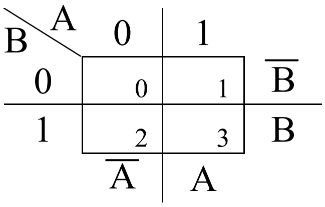
$=\{(X+\overline{Y})\cdot(X+Z)\}\cdot(X+Z)$

$=(X+\overline{Y})(X+Z)$

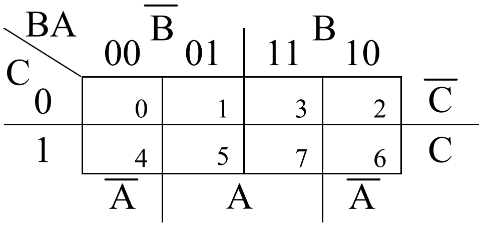
$=X+(\overline{Y}\cdot Z)$

KARNOUGH MAP (K-MAP):

1. Two variable B, A:



2. Three variable A, B, C:



3. Four variables:

		$\overline{A}$	A		$\overline{A}$	
	BA DC	00	01	11	10	
$\overline{D}$	00	0	1	3	2	$\overline{C}$
	01	4	5	7	6	C
D	11	12	13	15	14	
	10	8	9	11	10	$\overline{C}$
		$\overline{B}$		B		

يمكن تلخيص طريقة تبسيط للمعادلات باستخدام خارطة كار نوف وفق الأتي:

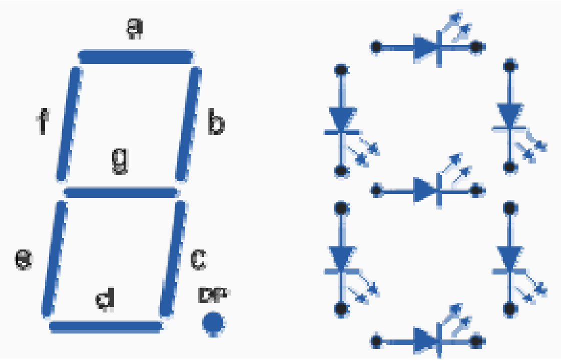
1. ثبت إل 1 على الخارطة لكل نتاج أساسي و املأ الشواغر بالا صفار أما الأعداد الممثلة بالمعادلـة والتـي غير مشمولة في التمثيل تمثل ب Don't Care (X).
2. حدد اكبر عدد من المربعات بأعداد 8 , 4 , 2 , أول (1) منعزل على نفسه من حالة عدم وجود عنصر أخر معه مع تذكر خاصيتي اللف والتشابك.
3. استعراض المجاميع واحذف كل مجموعة اشتركت جميع عناصرها مع مجموعات أخرى.
4. اكتب المعادلة البولينية جامعا الحدود ببوابة OR.
5. لا تعتبر التعابير المرادفة اختصارا.

Simplified of the O/P of 7 segment display

There are two important types of 7-segment LED digital display.

- The Common Cathode Display (CCD) - In the common cathode display, all the cathode connections of the LEDs are joined together to logic "0" and the individual segments are illuminated by application of a "HIGH", logic "1" signal to the individual Anode terminals.
-
- The Common Anode Display (CAD) - In the common anode display, all the anode connections of the LEDs are joined together to logic "1" and the individual segments are illuminated by connecting the individual Cathode terminals to a "LOW", logic "0" signal.

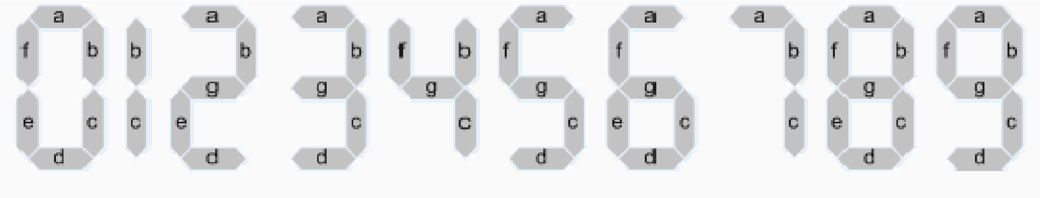
7-Segment Display Format



Truth Table for a 7-segment display

Individual Segments							Display
a	b	c	d	e	f	g	
×	×	×	×	×	×		0
	×	×					1
×	×		×	×		×	2
×	×	×	×			×	3
	×	×			×	×	4
×		×	×		×	×	5
×		×	×	×	×	×	6
×	×	×					7

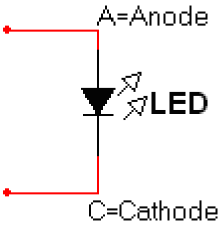
Individual Segments							Display
a	b	c	d	e	f	g	
×	×	×	×	×	×	×	8
×	×	×			×	×	9
×	×	×		×	×	×	A
		×	×	×	×	×	b
×			×	×	×		C
	×	×	×	×		×	d
×			×	×	×	×	E
×				×	×	×	F



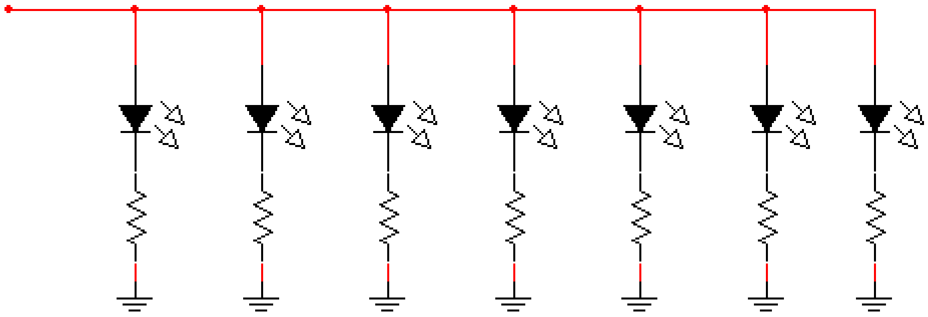
7-Segment Display Elements for all Numbers.

It can be seen that to display any single digit number from 0 to 9 or letter from A to F, we would need 7 separate segment connections plus one additional connection for the LED's "common" connection. Also as the segments are basically a standard light emitting diode, the driving circuit would need to produce up to 20mA of current to illuminate each individual segment and to display the number 8, all 7 segments would need to be lit resulting a total current of nearly 140mA, (8 x 20mA). Obviously, the use of so many connections and power consumption is impractical for some electronic or microprocessor based circuits and so in order to reduce the number of signal lines required to drive just one single display, display decoders such as the BCD to 7-Segment Display Decoder and Driver IC's are used instead.

LED (Lighte Emitting Diode) is the base of the 7-Segment wich are:

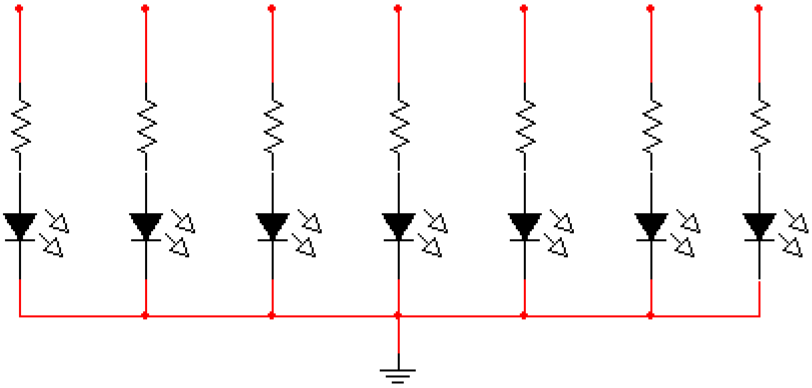


1. Common Anode.



(1)Common Anode

2. Common Cathode.



(2)Common Cathode

Exercise:

Simplifiy the O/P of 7-segment display as in page 12 ????

	D	C	B	A	G	F	E	D	C	B	A
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							
10	1	0	1	0							
11	1	0	1	1							
12	1	1	0	0							
13	1	1	0	1							
14	1	1	1	0							
15	1	1	1	1							

Designinig Combinatioal Logic Circuits

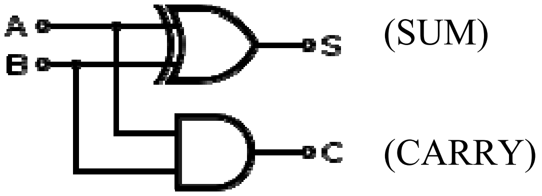
ADDERS & SUBTRUCTORS

The Half Adder

A **half adder** is a logical circuit that performs an addition operation on two one-bit binary numbers. The half adder outputs a sum of the two inputs and a carry value.

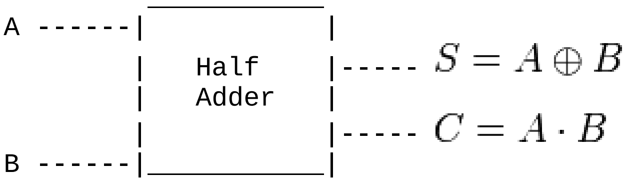
The drawback of this circuit is that in case of a multibit addition, it cannot include a carry.

$$S = A \oplus B$$
$$C = A \cdot B$$



Truth Table

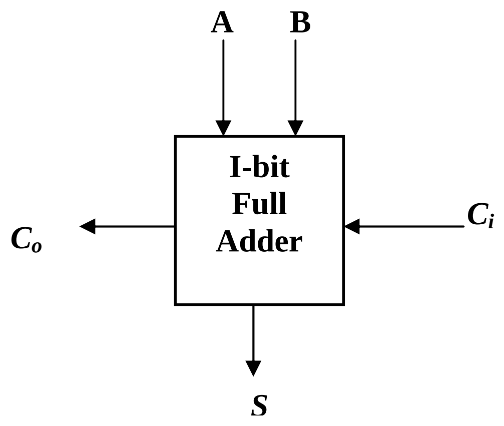
Input (I/P)		Output (O/P)	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1



Schematic Symbol of Half Adder

The Full Adder

A **full adder** is a logical circuit that performs an addition operation on three one-bit binary numbers. The full adder produces a sum of the two inputs and carry value. It can be combined with other full adders (see below) or work on its own.



Schematic symbol for a 1-bit full adder

Truth Table

Input (I/P)			Output (O/P)	
A	***B***	***C_i***	***S***	***C_o***
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

S = AB’Ci’ +A’BCi’ +A’B’Ci +ABCi

Co = ABCi’+ AB’Ci + A’BCi + ABCi

S = Ci’(AB’ + A’B) + Ci (A’B’+ AB)

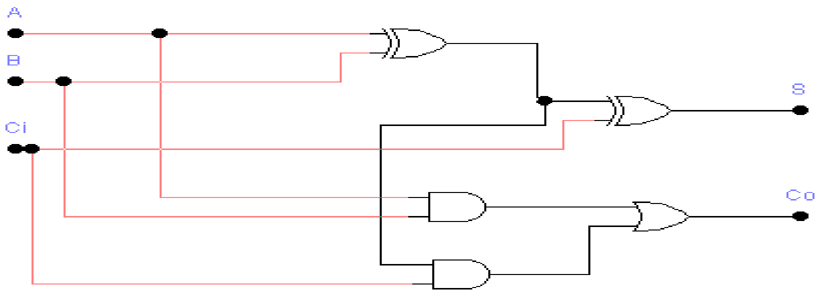
S = $Ci'(A \oplus B) + Ci\,\overline{(A \oplus B)}$

S = $Ci \oplus (A \oplus B)$ ………………………..…………………(1)

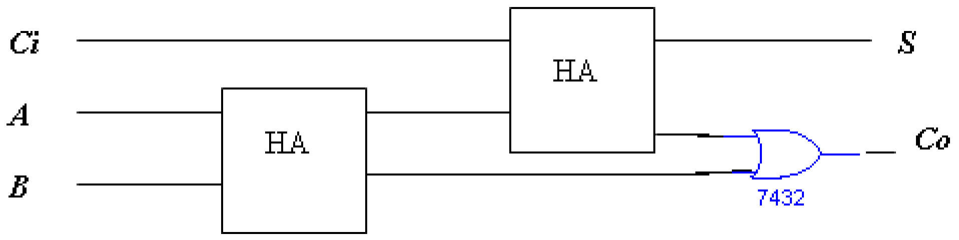
Co = AB(Ci’+ Ci) + Ci(B’A +A’B)

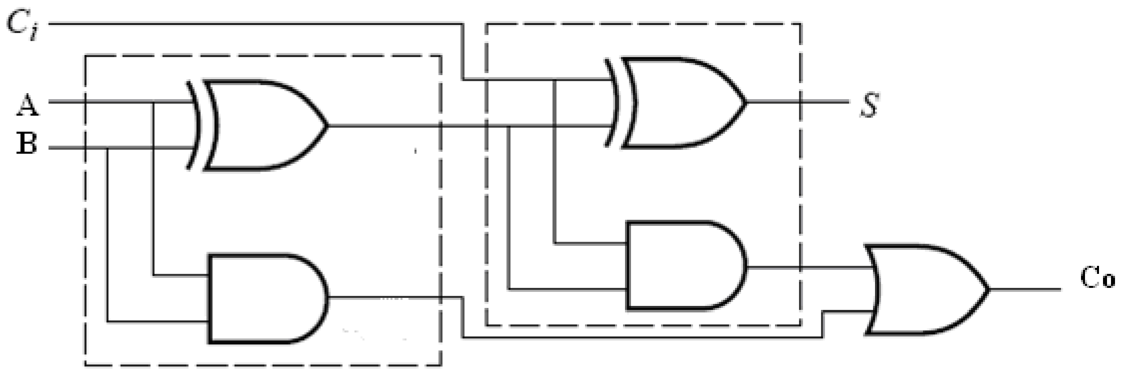
Co = $AB + Ci(A \oplus B)$ …………………… ………………..(2)

From equations 1 and 2 , we can draw the following cct. .

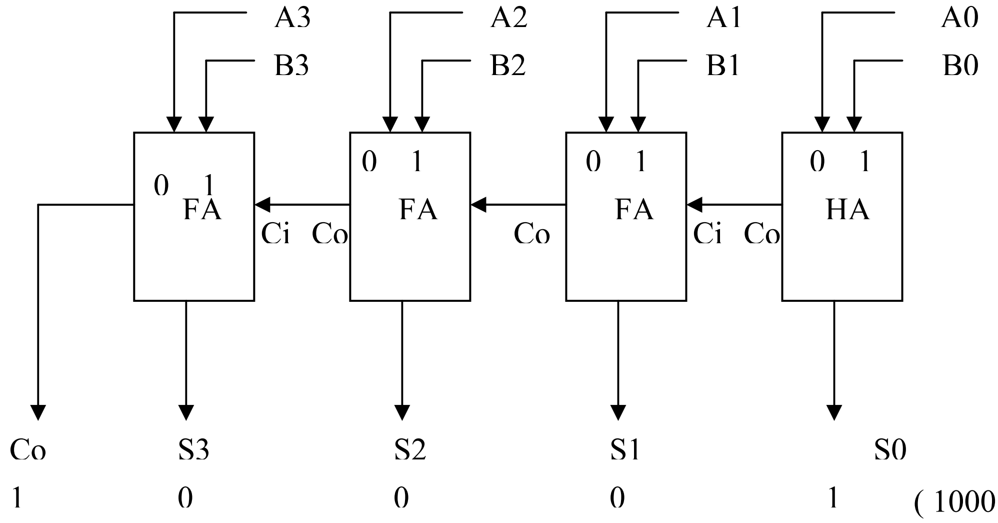


FA also can be implemented from two HA and the following figures explain this.





Parallel Binary Adder



A3 A2 A1 A0
+ B3 B2 B1 B0

S3 S2 S1 S0

1 0 0 0   8
+ 1 0 0 1   9
1 0 0 0 1   17

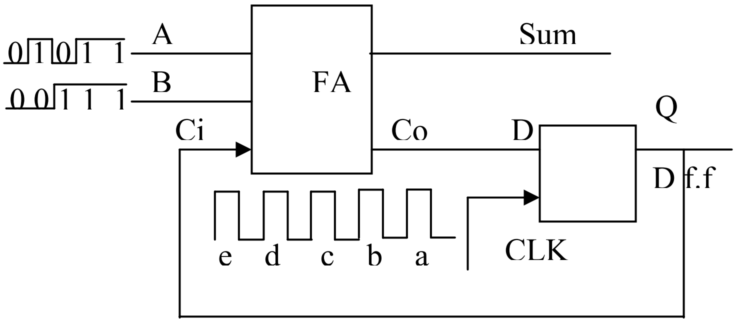
(10001) 2 = (17) 10

Note : we can replace the FH by FA after making Ci = 0 for it equal .
The No. of Adders = No. of bit , therefore , the parallel adder is fast but complex in implementation.

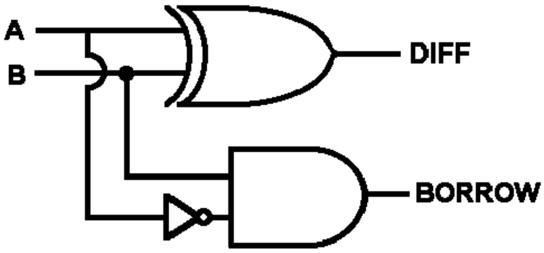
Serial Binary Addition

Serial performs its addition, it is partially dependent on the clock cycle therefore it is slower than Parallel adder but less complexity. Serial Binary Adder uses one FA , one Df.f and 3 Registers which requires number of clocks that’s equaled to the number of bits. The important things is the synchronization between clocks and addition the numbers.

The following figures explain it .



Half Binary SUBTRACTeR



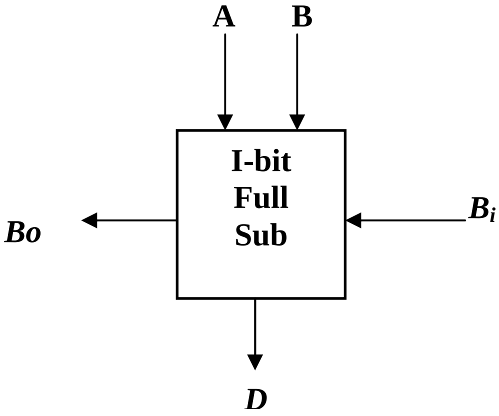
0 – 0 = 0 0
1 – 0 = 1 0
1 – 1 = 0 0
0 – 1 = 1 1

Truth Table

Input (I/P)		Output (O/P)	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	1
1	1	0	0

```
          ____________
A ------|            |
        |   HALF     |----- S = A ⊕ B
        |   SUB      |
        |   ( HS )   |----- C = A’. C
B ------|____________|
```

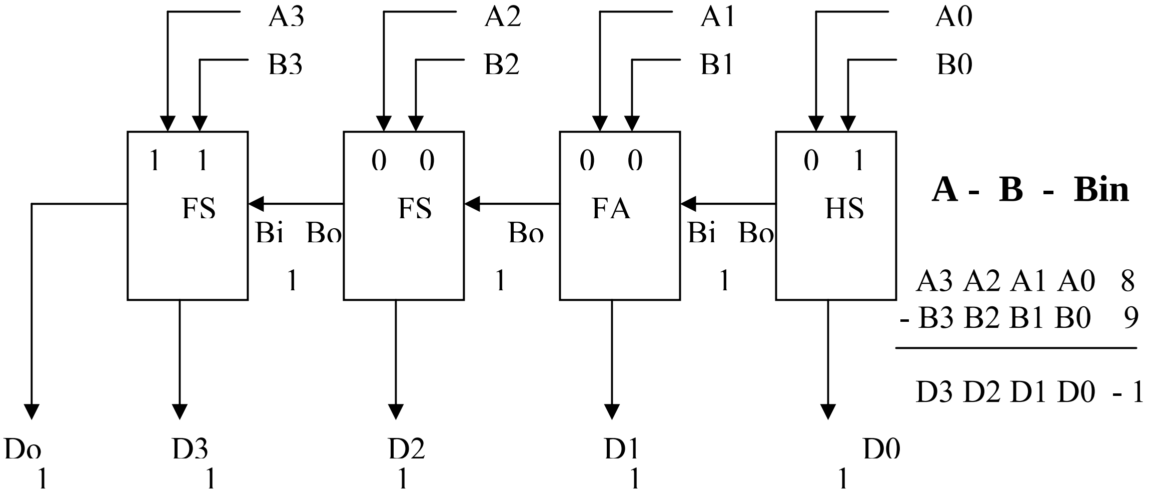
Full Binary Subtracter



Truth Table

Input (I/P)			Output (O/P)	
A	***B***	***B_i***	***D***	***B_o***
0	0	0	0	0
1	0	0	1	0
0	1	0	1	1
1	1	0	0	0
0	0	1	1	1
1	0	1	0	0
0	1	1	1	1
1	1	1	1	1

Parallel Binary Subtracter



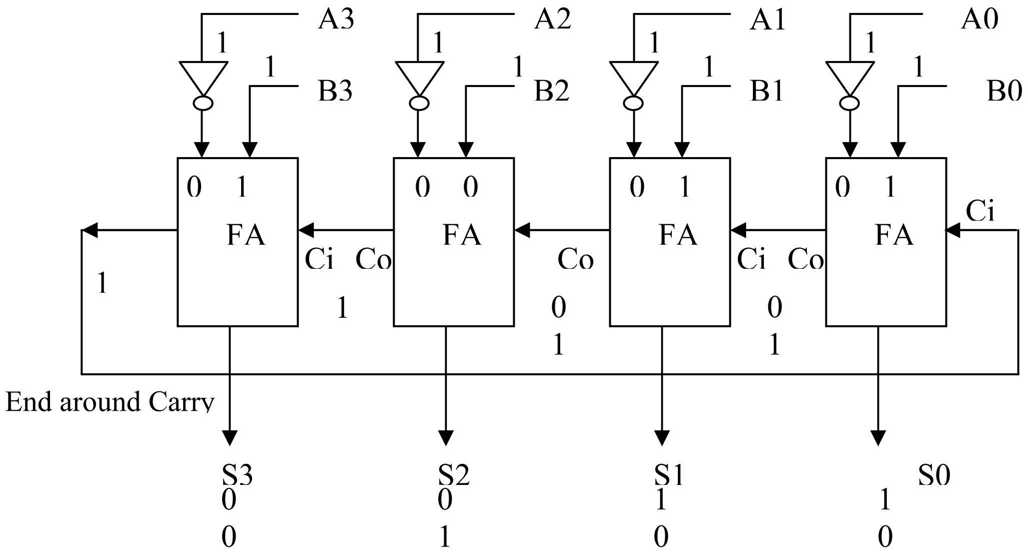
4 – bit parallel Binary Subtractor

The circuit give us direct result when **A >B** but when **A < B** we have to take the 2nd complement for the result.

Subtraction by using Adder

There are 2 types for subtracting process . Both of them are used FA and Inverter to performer the subtraction A- B = A + (- B)

1. **1S complement subtracter (End around carry).**
2. **2S complement subtracter.**

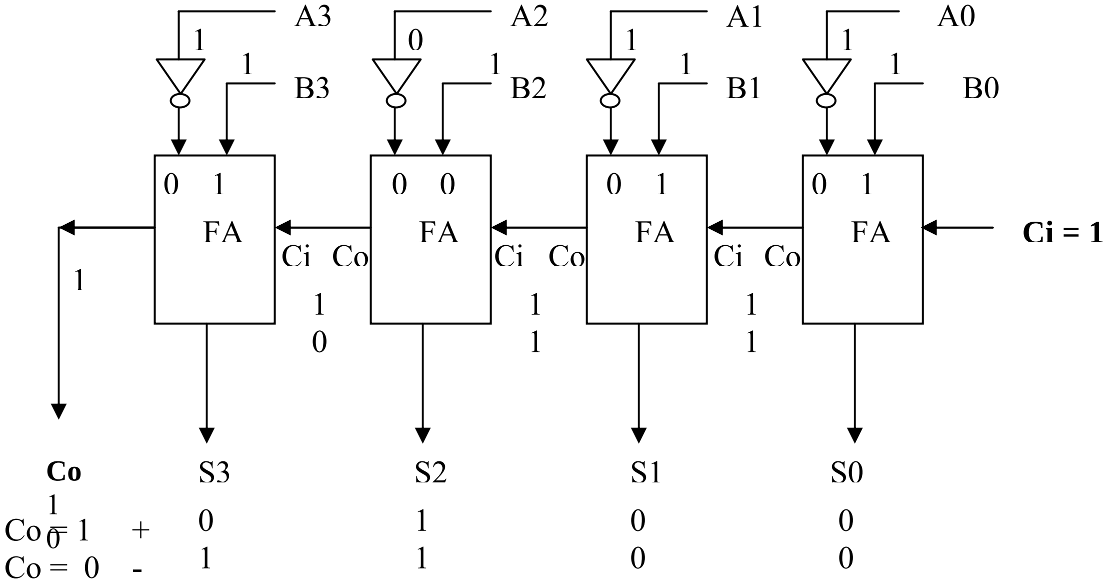


4- bit 1S complement Binary Subtraction (End around Carry)

Example:

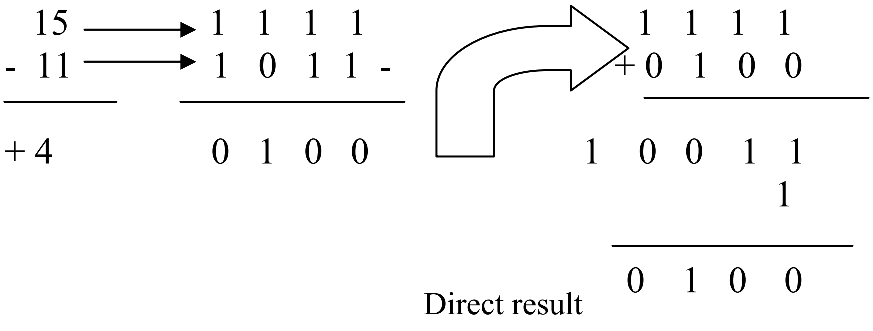
```
  15 ──────► 1 1 1 1                 1 1 1 1
- 11 ──────► 1 0 1 1 -             + 0 1 0 0
──────      ─────────────          ───────────
+ 4          0 1 0 0               1 0 0 1 1
                                           1
                                   ───────────
                                     0 1 0 0
          Positive sign
```

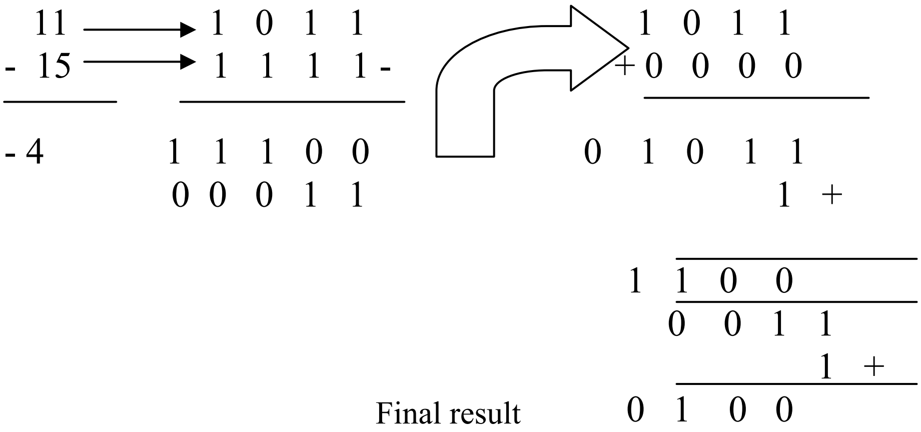
```
  11 ──────► 1 0 1 1                 1 0 1 1
- 15 ──────► 1 1 1 1 -             + 0 0 0 0
──────      ─────────────          ───────────
- 4        1 1 1 0 0               0 1 0 1 1
           0 0 0 1 1

                                   ───────────
                                     1 0 1 1
        Negative sign
                                     0 1 0 0
```



4 – bit 2S complement subtraction

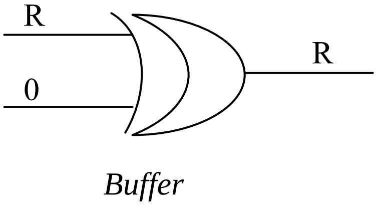
Example:

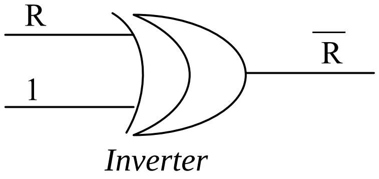


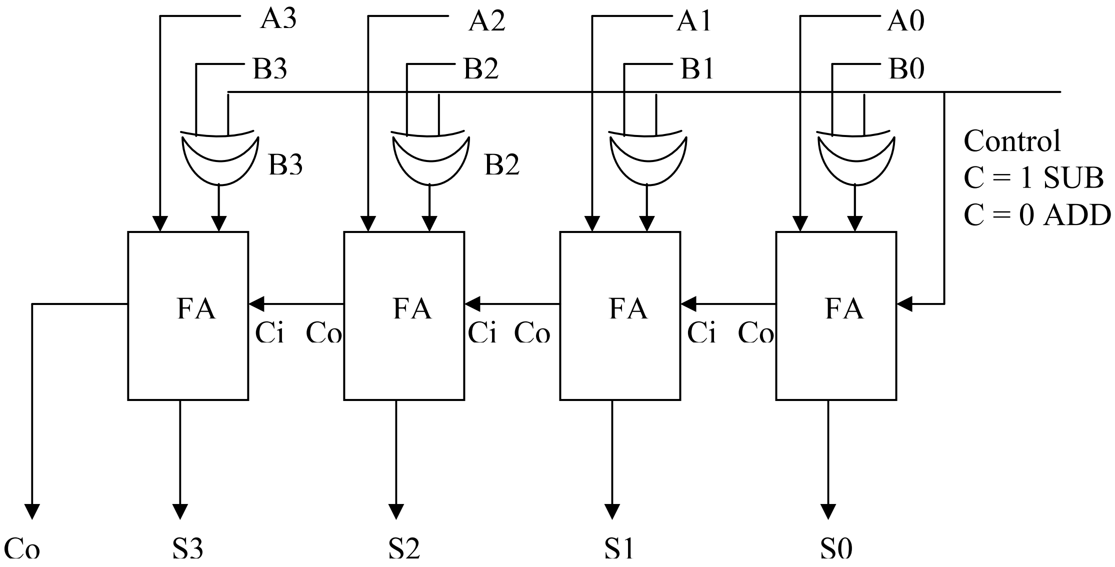


To perform the addition and subtraction Processes , It will be better to do it in the same circuit by using simple control gates (Ex-OR) .

A - B = A + (- B) = A + (B' + 1)







2S Complement Adder- Subtrater

BCD Addition

BCD, or *binary-coded decimal*, represents the 10 decimal digits in terms of binary numbers. It is possible to build digital hardware that man-ip-ulates BCD directly, and such hardware could be found in early com-puters and many hand-held calculators. The BCD system was chosen for the internal number system in these machines because it is easy to convert it to alphanumeric representations for printouts and displays. The compelling advantages of BCD have waned over time, and these digits are supported by more modern hardware simply to provide backward compatibility with earlier generations of machines. In this section, we briefly examine the approaches for constructing BCD arithmetic -elements.

BCD Number Representation

We have met BCD representation in the previous lectures. The decimal digits 0 through 9 are represented by the 4-bit binary strings 0000 through 1001. The remaining 4-bit encodings, (1010)2 through (1111)2, are treated as don't cares.

Just as in conventional decimal addition, BCD addition is performed one decimal digit at a time. The question is, what happens when the sum exceeds what can be represented in 4 bits? Stated differently, what are the conditions under which a carry is generated to the next highest-order BCD digit?

For example, let's consider the addition of the two BCD digits 5 and 3:

```
5 =  0101
3 =  0011
     ----
     1000 = 8
```

Now consider the sum of 5 and 8:

```
5 =  0101
8 =  1000
     ----
     1101 = 13!
```

The sum is (1101)2 = 13, but this result should be correctly represented as 0001 0011 in BCD notation. Fortunately, there is a simple way to find the correct result. We add 6 (01102) to the digit sum if it exceeds 9. Let's examine the following cases:

```
5 =  0101                         9 =  1001
8 =  1000                         7 =  0111
     ----                              ------
     1101 = 13 in decimal              1 0000 = 16 in decimal
    + 0110                             + 0110
    ------                             ------
    1 0011 = 1 3 in BCD                1 0110 = 1 6 in BCD
```

In both cases, by adding six we obtain the correct answer in BCD. This observation is critical to the design of a BCD adder, as we shall see in the next subsection.

BCD Adder Design

Figure below gives a block diagram implementation for a BCD adder. The first row of full adders implements a conventional 4-bit binary adder. The second row provides the capability to add 01102 when the sum obtained by the first row exceeds 9 (1001)2.

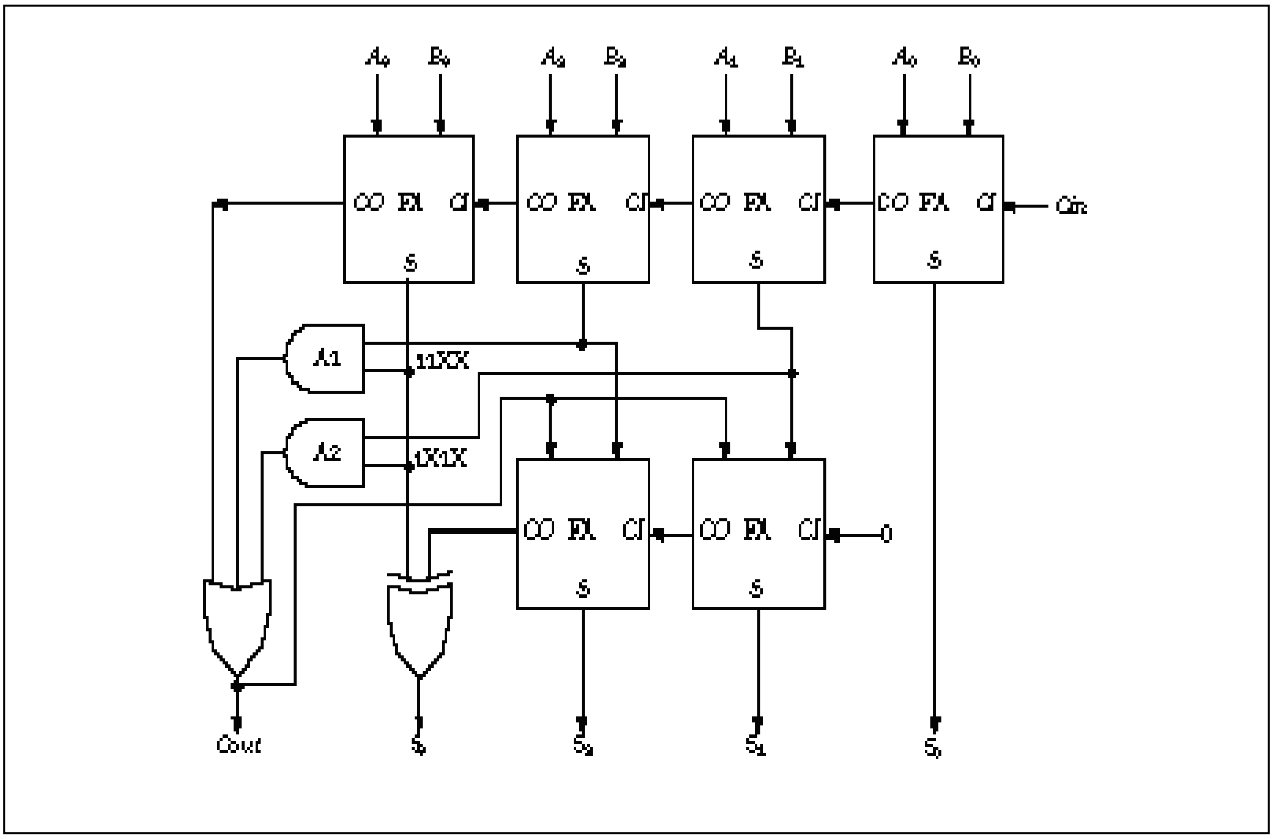
Here is how it works. The adders of the second row add the carry-out bit to the sum bits *S*2 and *S*1. Carry-out should be asserted in cases in which we need to add the correction factor. What are these cases?

The AND gates labeled *A*1 and *A*2 detect the conditions under which the first-level sum matches the patterns 11XX2 and 1X1X2. These are exactly the cases in which this sum exceeds 9. When carry-out is asserted, the XOR gate and the adders in the second row effectively add (0110)2 to the first row's sum.

There is one further case to consider. The correction factor should also be applied whenever the first-row sum exceeds 15. We saw such an example with the sum of 9 and 7 above. This case is easy to detect: the carry-out of the first-row adders will be asserted.

Thus the sum exceeds 9 if either the first-row carry-out is asserted, or the sum matches the pattern 11XX2, or the sum matches the pattern 1X1X2. These are precisely the inputs to the OR gate that computes the BCD carry-out.

A BCD adder requires over 50% more hardware than a comparable binary adder. Since faster binary adders are now available, it is no surprise that they have replaced BCD adders in almost all applications.

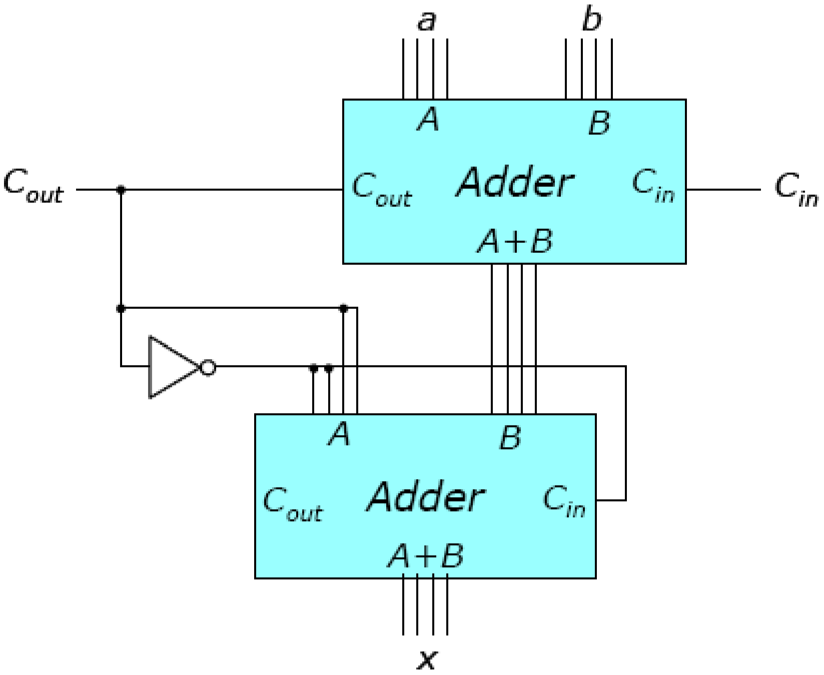


Excess-3 Adder Design

When you add two XS-3 numbers together, the result is not an XS-3 number. For instance, when you add 1 and 0 in XS-3 the answer seems to be 4 instead of 1. In order to correct this problem, when you are finished adding each digit, you have to subtract 3 (binary 11) if the digit is less than decimal 10 and add three if the number is greater than or equal to decimal 10 (thus causing the number to wrap).

Your circuit will have two sets of four inputs *a*=*a*3,*a*2,*a*1,*a*0 and *b*=*b*3,*b*2,*b*1,*b*0. It will also have four outputs *x*=*x*3,*x*2,*x*1,*x*0. The output *x* of your circuit should be the excess-3 sum of the input values *a* and *b*. So for example, if *a*=0100 (representing the value 1) and *b*=1000 (representing the value 5), the output *x*=1001 (representing the value 6).
Your circuit should correctly wrap-around if the two input values are too large. So for example, if *a*=0111 (representing the value 4) and *b*=1011 (representing the value 8), the output *x*=0101 (representing the value 2). Your circuit should also have a carry input *Cin* and a carry output *Cout*.

A block diagram of a circuit that implements the single digit BCD adder is shown below. The two large blocks are ordinary 4 bit *binary* adders. In your design notes, include an explanation for why this design produces the correct excess-3 sum and the correct value for *Cout*. Use the schematic editor to create a schematic for this circuit and simulate it. You may use the four bit adder component in the schematic editor's symbol library (you'll find it in the arithmetic section of the library). Ignore the OFL output (this is used when doing signed arithmetic). Since the circuit is too large for exhaustive testing, select test cases that demonstrate that the circuit works correctly and include an explanation for why these test cases are sufficient. Include "boundary cases" such as adding 0 or 1 and input combinations that are just large enough to generate a carry. Turn in a copy of your design notes, the schematic and the simulation results.

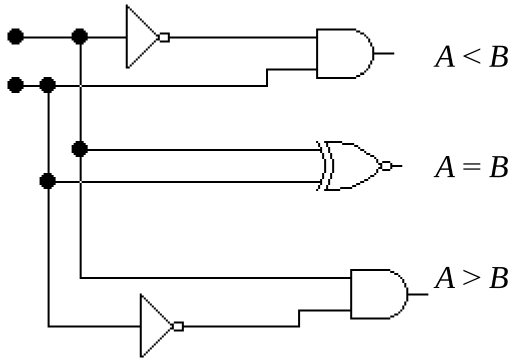


Digital comparator

A **digital comparator** or **magnitude comparator** is a hardware electronic device that takes two numbers as input in binary form and determines whether one number is greater than, less than or equal to the other number. Comparators are used in a central processing units (CPU) and microcontrollers. Examples of digital comparator include the CMOS 4063 and 4585 and the TTL 7485 and 74682-'89.

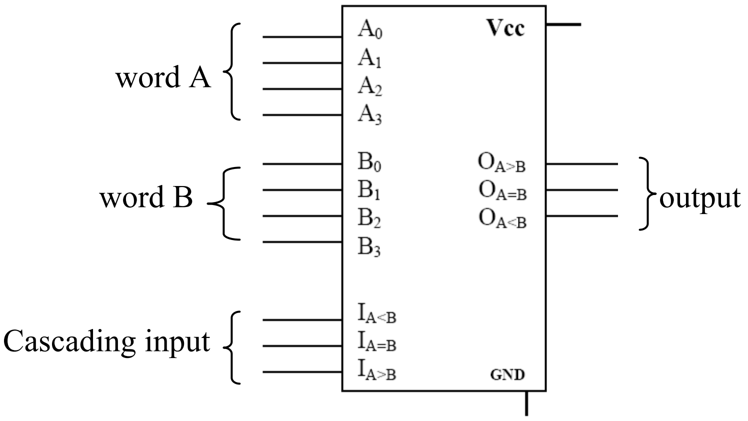
The analog equivalent of digital comparator is the voltage comparator. Many microcontrollers have analog comparators on some of their inputs that can be read or trigger an interrupt.

1-bit Comparator



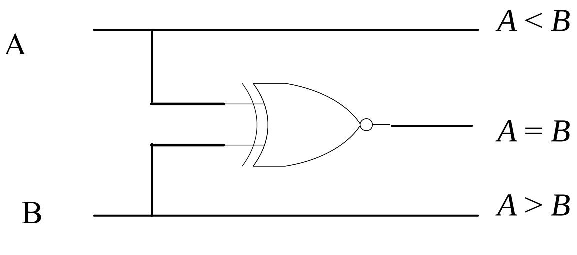
Truth Table

Input (I/P)		Output (O/P)		
A	B	$A > B$	$A = B$	$A < B$
0	0	0	1	0
1	0	1	0	0
0	1	0	0	1
1	1	0	1	0



Block diagram of IC 7485 as 4- bit digital comparator

You can connect the 1 bit digital comparator to cascade input of IC 7485 to get 5- bit digital Comparator or simply to connect the following part to reach the same result.



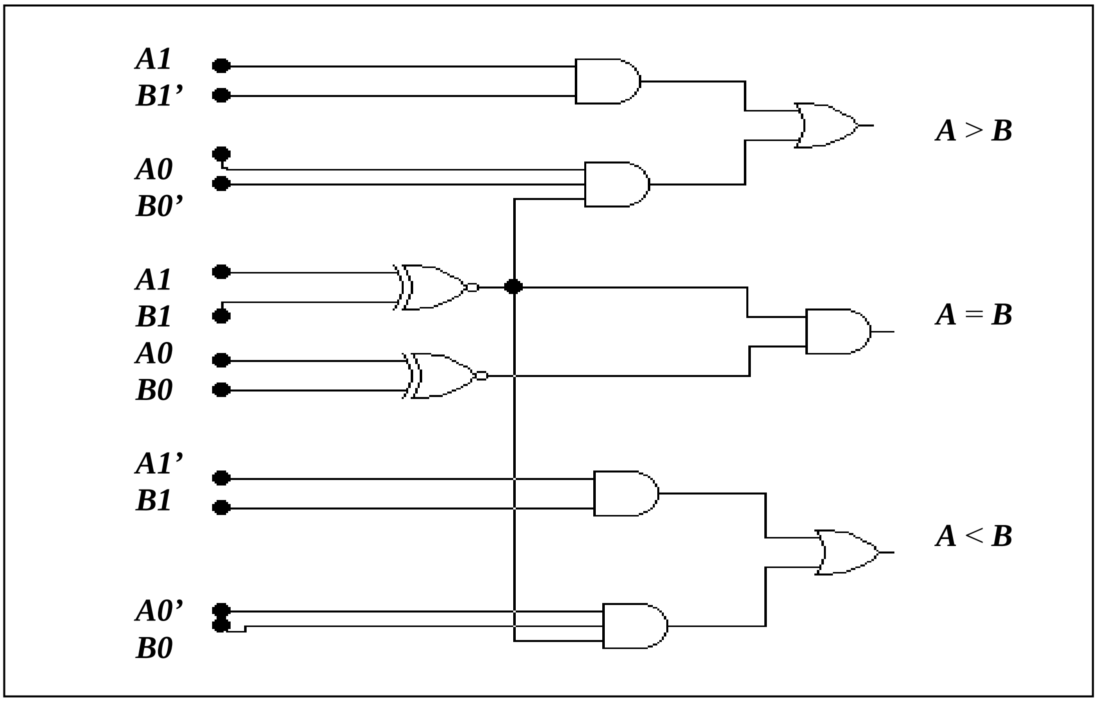
To design 2- bit digital comparator by using standard logic gates. Let us assume

A = A1A0 and B = B1B0.

A > ***B***= A1 > B1 **OR** A0 > B0 **AND** A1= B1.
$\boldsymbol{A} > \boldsymbol{B}=$ $A1\,B1' + (\,A0\,B0')\,,\,(A1B1 + A1'B1')$
$\boldsymbol{A} > \boldsymbol{B}=$ $A1\,B1' + (\,A0\,B0')\overline{(A1 \oplus B1)}$

A < ***B***= A1 < B1 **OR** A0 < B0 **AND** A1= B1.
$\boldsymbol{A} < \boldsymbol{B}=$ $A1'\,B1 + (\,A0'\,B0)\,,\,(A1B1 + A1'B1')$
$\boldsymbol{A} < \boldsymbol{B}=$ $A1'\,B1 + (\,A0'\,B0')\overline{(A1 \oplus B1)}$

A = ***B***= A1 = B1 **AND** A0 = B0
$\boldsymbol{A} = \boldsymbol{B}=$ $(A1B1 + A1'B1')(A1B1 + A1'B1')$
$\boldsymbol{A} = \boldsymbol{B}=$ $\overline{(A1 \oplus B1)}\;\overline{(A0 \oplus B0)}$



2-bit digital comparator

Not : You can test your design by assuming different values for both A word and b word.

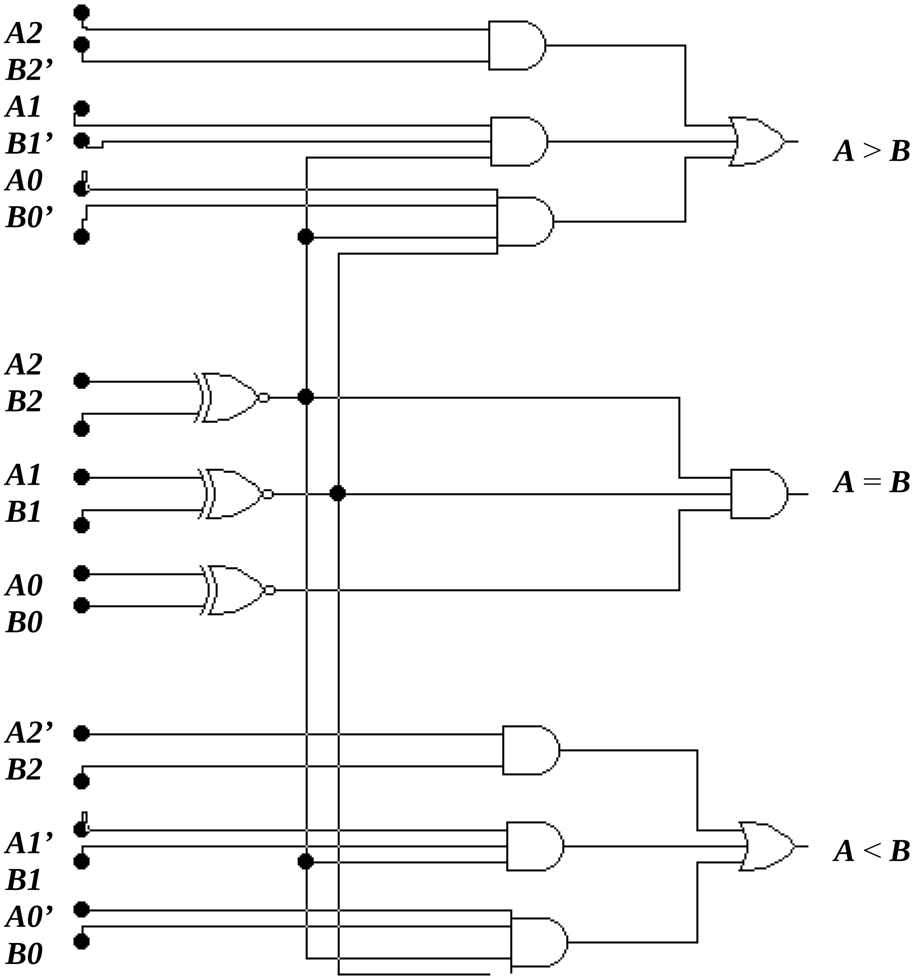
H.W.)) Design 3- bit digital comparator and 4- bit digital comparator by using simple gates.

To design 3- bit digital comparator by using standard logic gates. Let us assume
A = A2A1A0 and B = B2B1B0.

A > ***B*****=** A2 > B2 **OR** A1 > B1 **AND** (A2= B2) **OR** A0 > B0 **AND** (A1= B1) **AND** (A2= B2)..
A > ***B*****=** A2 B2’ + A1B1’$(\overline{A2 \oplus B2})$ + A0B0’$(\overline{A1 \oplus B1})$ $(\overline{A2 \oplus B2})$

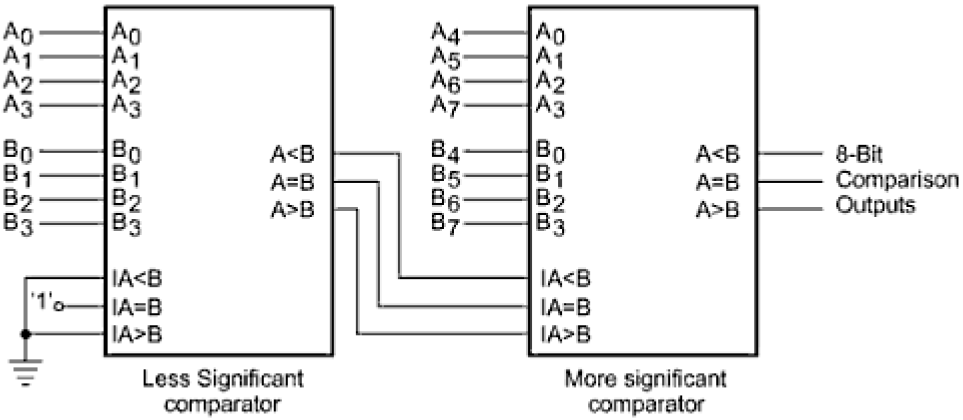
A = ***B*****=** **(** A2 = B2) **AND** (A1 = B1) **AND (** A0 = B0)
A = ***B*****=** $\overline{(A2 \oplus B2)\ (A1 \oplus B1)\ (A0 \oplus B0)}$

A > ***B*****=** A2 > B2 **OR** A1 > B1 **AND** (A2= B2) **OR** A0 > B0 **AND** (A1= B1) **AND** (A2= B2)..
A > ***B*****=** A2’ B2 + A1’B1$(\overline{A2 \oplus B2})$ + A0’B0$(\overline{A1 \oplus B1})$ $(\overline{A2 \oplus B2})$



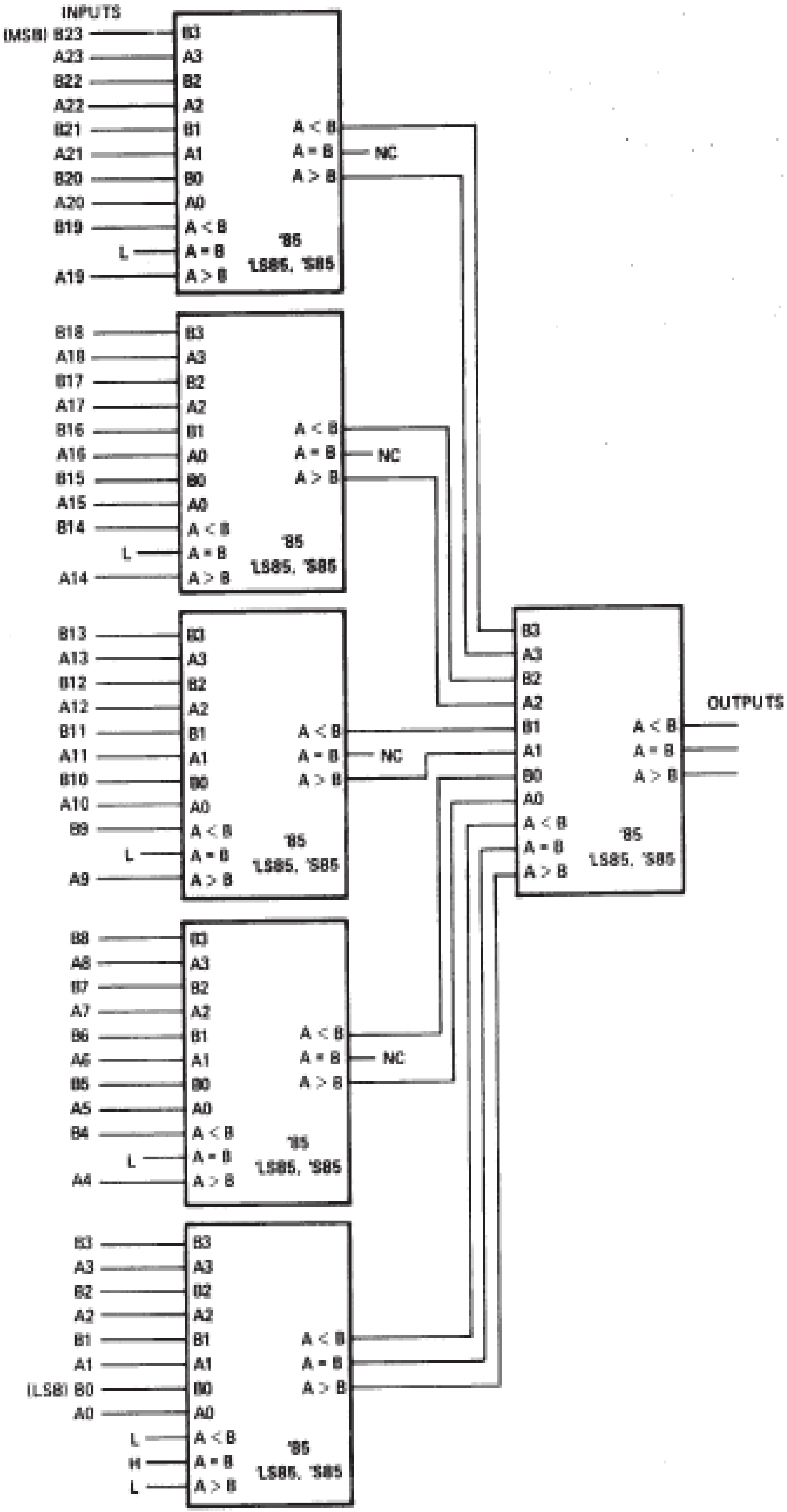
3-bit digital comparator

To use two 7485 IC to design a combinational circuit that compares two eight-bit numbers,
A = $A_7A_6A_5A_4A_3A_2A_1A_0$ and B = $B_7B_6B_5B_4B_3B_2B_1B_0$.
Note that the circuit number **1** compares the four least significant bits (0 to 3) and the circuit number **2** compares the four most significant inputs (4 to 7).



Design a two-bit magnitude comparator.

H.W)) Show how you can design high speed method of comparing two 24- bit words with only two levels of device delay? Use sex 7485 that you can connect them in parallel .


INPUTS
(MSB) B23
A23
B22
A22
B21
A21
B20
A20
B19
L
A19
B3
A3
B2
A2
B1
A1
B0
A0
A < B
A = B
A > B
'85
'LS85, 'S85
A < B
A = B
A > B
NC
B18
A18
B17
A17
B16
A16
B15
A15
B14
L
A14
B3
A3
B2
A2
B1
A0
B0
A0
A < B
A = B
A > B
'85
'LS85, 'S85
NC
B13
A13
B12
A12
B11
A11
B10
A10
B9
L
A9
'85
'LS85, 'S85
NC
B8
A8
B7
A7
B6
A6
B5
A5
B4
L
A4
'85
'LS85, 'S85
NC
B3
A3
B2
A2
B1
A1
(LSB) B0
A0
L
H
L
'85
'LS85, 'S85
B3
A3
B2
A2
B1
A1
B0
A0
A < B
A = B
A > B
'85
'LS85, 'S85
OUTPUTS
A < B
A = B
A > B

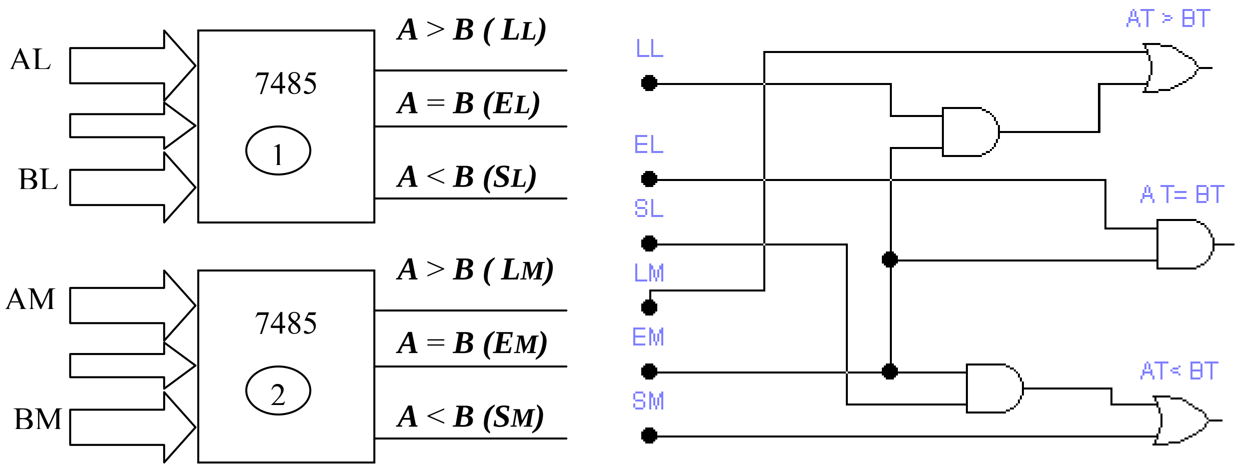
To compare large numbers by using only 2 ICs 7485, this will be easy to consider the numbers as blocks.
Ex. To compare 3417 with 7883 as decimals numbers, we can do the following blocks:

	M	L
3131	31	31
3017	30	17

$\boldsymbol{A_T > B_T} =$ $A_M > B_M + (A_M{=}B_M)\ A_L > B_L$
$= \boldsymbol{L_M + (E_M)\ L_L}$

$\boldsymbol{A_T = B_T} =$ $(A_M = B_M)$ AND $(A_M{=}B_M)$
$= \boldsymbol{E_M\ E_L}$

$\boldsymbol{A_T < B_T} =$ $A_M < B_M + (A_M{=}B_M)\ A_L < B_L$
$= \boldsymbol{S_M + (E_M)\ S_L}$



To compare a number with its sign by using IC 7485 with some gates,, this will be easy to deal with sign as extra signal.
The general design will be:

A3	A2	A1	A0	sign

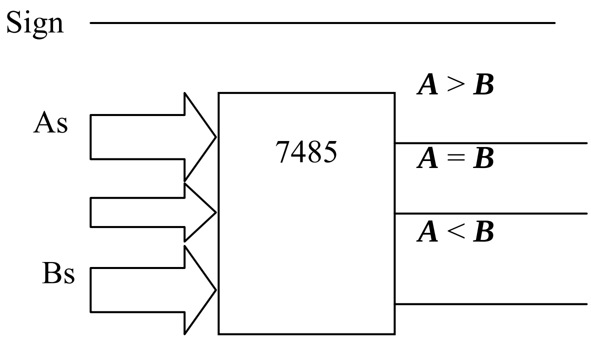
When sign flag = 0 +signal
When sign flag = 1 -signal

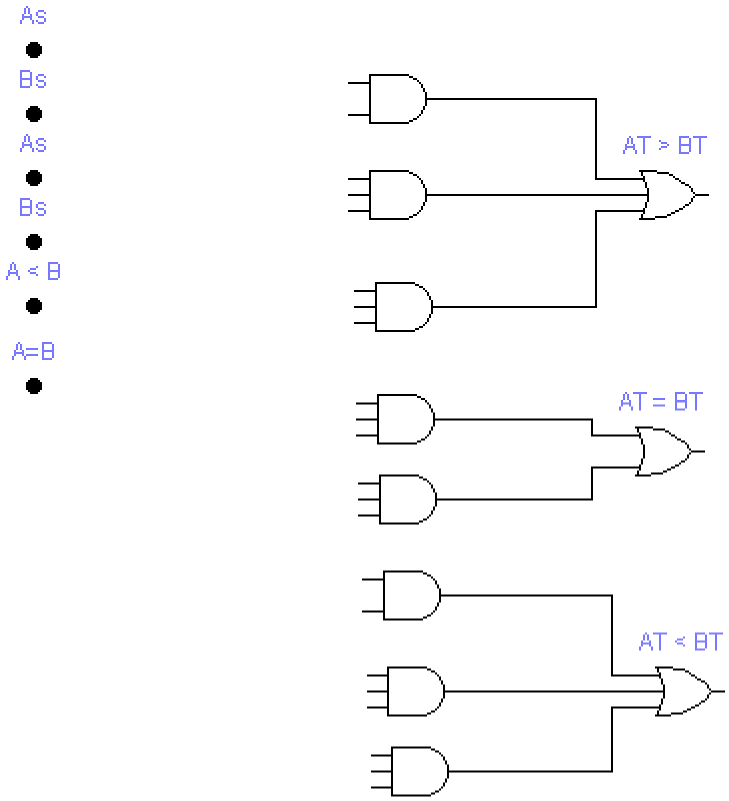
B3	B2	B1	B0	sign

$\boldsymbol{A_T > B_T} = \bar{A}_s B_s + \bar{A}_s\bar{B}_s\,(A > B) + A_sB_s(B > A)$

$\boldsymbol{A_T < B_T} = A_s \bar{B}_s + \bar{A}_s\bar{B}_s\,(A < B) + A_sB_s(B < A)$

$\boldsymbol{A_T = B_T} = A_sB_s\,(A{=}B) + \bar{A}_s\bar{B}_s\,(A{=}B)$

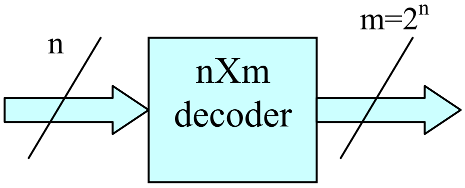

Sign
As
Bs
7485
A > B
A = B
A < B


As
Bs
As
Bs
A < B
A=B
AT > BT
AT = BT
AT < BT

Encoder And Decoder

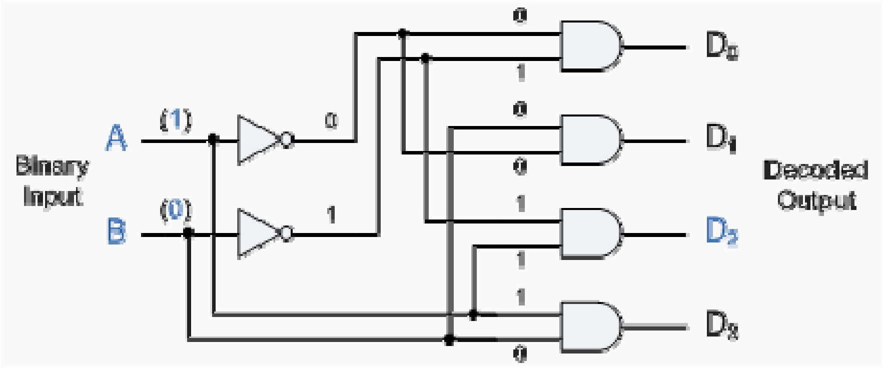
Binary Decoders

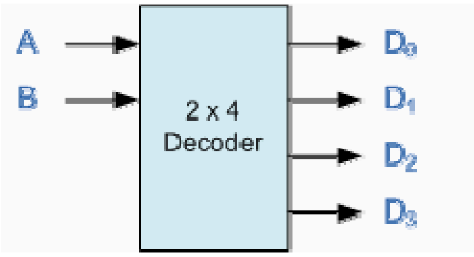
A **Decoder** is the exact opposite to that of an "Encoder". It is basically, a combinational type logic circuit that converts the binary code data at its input into one of a number of different output lines, one at a time producing an equivalent decimal code at its output. **Binary Decoders** have inputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines, and a **"n-bit"** decoder has **2^n** output lines. Typical combinations of decoders include, 2-to-4, 3-to-8 and 4-to-16 line configurations. Binary Decoders are available to "decode" either a Binary or BCD input pattern to typically a Decimal output code.

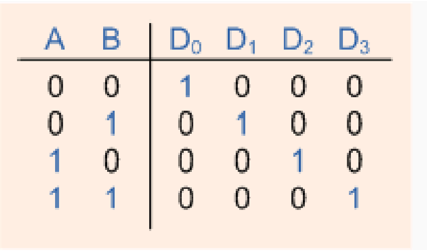


The decoder is called n to m
Where m ≤ 2^n

Example: A 2-to-4 Binary Decoders.



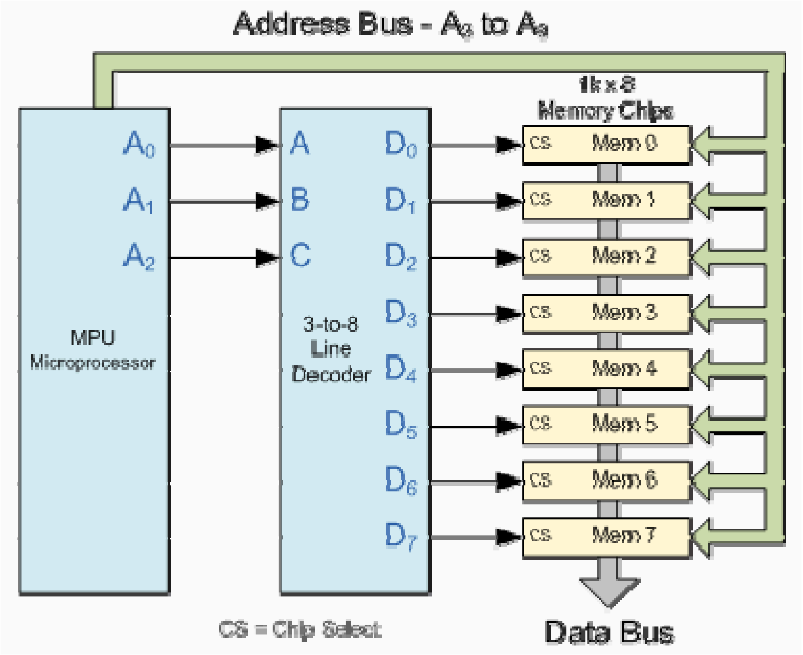




A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

In this simple example of a 2-to-4 line binary decoder, the binary inputs A and B determine which output line from D0 to D3 is "HIGH" at logic level "1" while the remaining outputs are held "LOW" at logic "0". Therefore, whichever output line is "HIGH" identifies the binary code present at the input, in other words it "de-codes" the binary input and these types of binary decoders are commonly used as **Address Decoders** in microprocessor memory applications.

Memory Address Decoding.

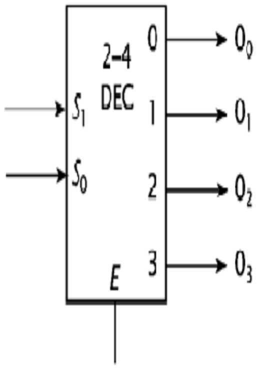


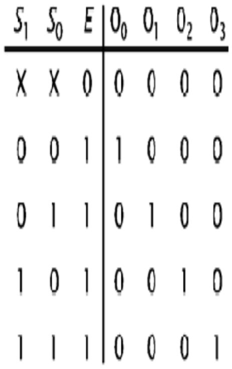
The binary decoder requires 3 address lines, (A_0 to A_2) to select each one of the 8 chips (the lower part of the address), while the remaining 7 address lines (A_3 to A_9) select the correct memory location on that chip (the upper part of the address). Having selected a memory location using the address bus, the information at the particular internal memory location is sent to the "Data Bus" for use by the microprocessor. This is of course a simple example but the principals remain the same for all types of memory chips or modules.

Expansion of Decoder

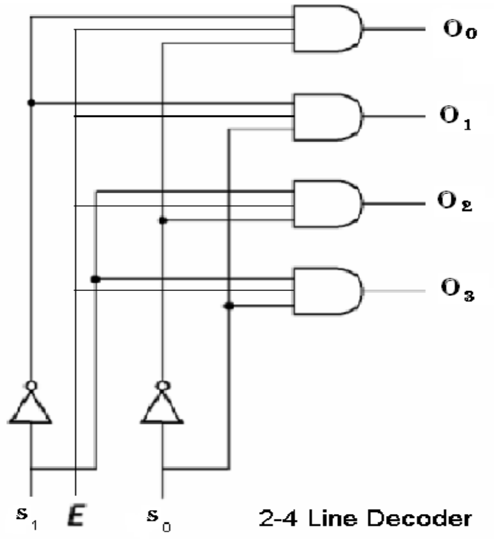
The expansion of Decoder is achieved using decoder having enable control, as shown below.

Example : 2-to-4 Decoder with enable (E).

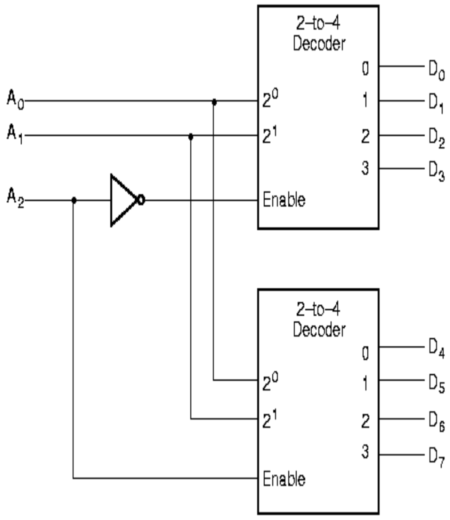


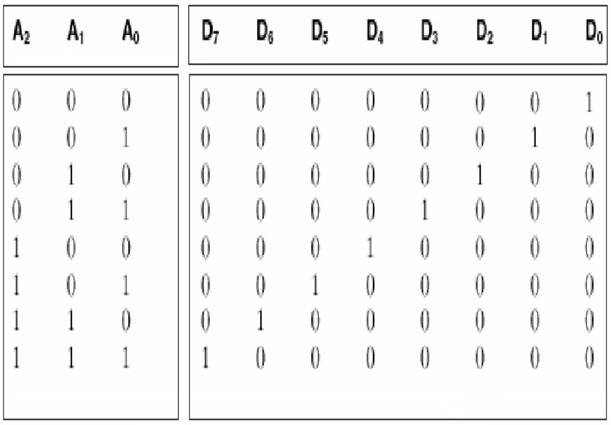


S_1	S_0	E	O_0	O_1	O_2	O_3
X	X	0	0	0	0	0
0	0	1	1	0	0	0
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1



Example : Design 3X8 Decoder from two 2X4 Decoder with Enable.



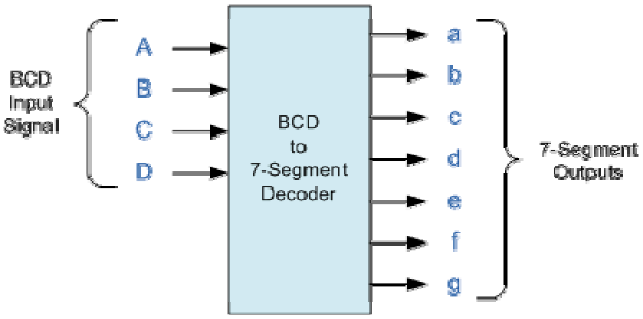


A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

H.W)) design 4X16 from 3X8 Decoder.

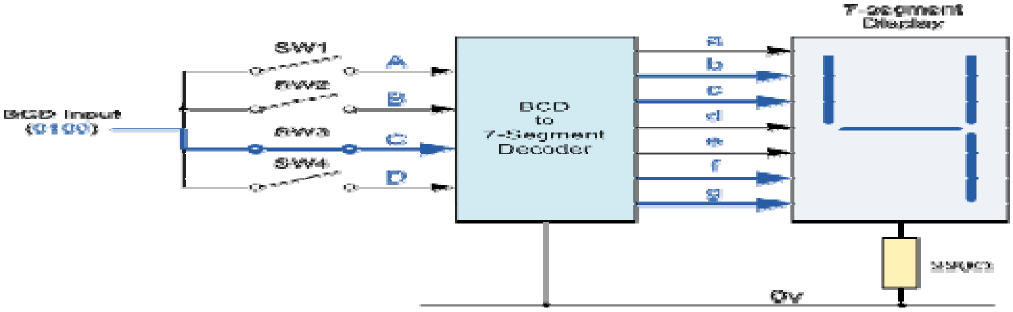
Binary Decoders are very useful devices for converting one digital format to another, such as binary or BCD type data into decimal or octal etc and commonly available decoder IC's are the TTL 74LS138 3-to-8 line binary decoder or the 74ALS154 4-to-16 line decoder. They are also very useful for interfacing to 7-segment displays such as the TTL 74LS47.

BCD to 7-Segment Decoder



The use of **packed** BCD allows two BCD digits to be stored within a single byte (8-bits) of data, allowing a single data byte to hold a BCD number in the range of 00 to 99.

An example of the 4-bit BCD input (0100) representing the number 4 is given below.

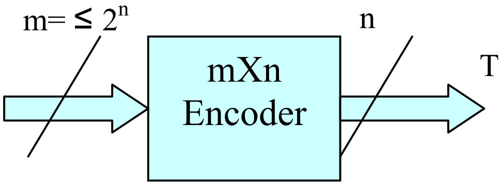


In practice current limiting resistors of about 150Ω to 220Ω would be connected in series between the decoder/driver chip and each LED display segment to limit the maximum current flow. Different display decoders or drivers are available for the different types of display available, e.g. 74LS48 for common-cathode LED types, 74LS47 for common-anode LED types, or the CMOS CD4543 for liquid crystal display (LCD) types.

Liquid crystal displays (LCD´s) have one major advantage over similar LED types in that they consume much less power and nowadays, both LCD and LED displays are combined together to form larger Dot-Matrix Alphanumeric type displays.

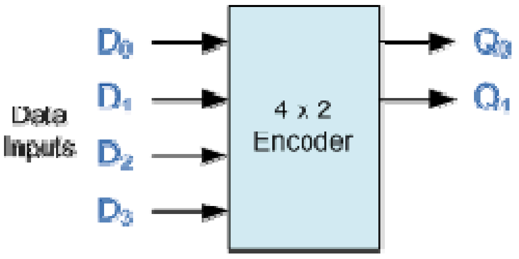
The Encoder

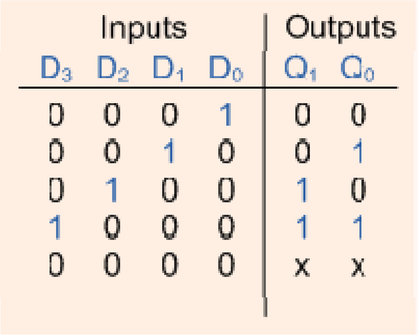
Unlike a multiplexer that selects one individual data input line and then sends that data to a single output line or switch, an **Encoder** takes all the data inputs one at a time and converts them to a single encoded output. Then, it is a multi-input data line, combinational logic circuit that converts the logic level "1" data at its inputs to an equivalent binary code at its output. Generally encoders produce outputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines and a "n-bit" encoder has 2^n input lines with common types that include 4-to-2, 8-to-3 and 16-to-4 line configurations. Encoders are available to encode either a decimal or hexadecimal input pattern to typically a binary or B.C.D. output code.



The encoder is called m to n

4-to-2 Bit Encoder





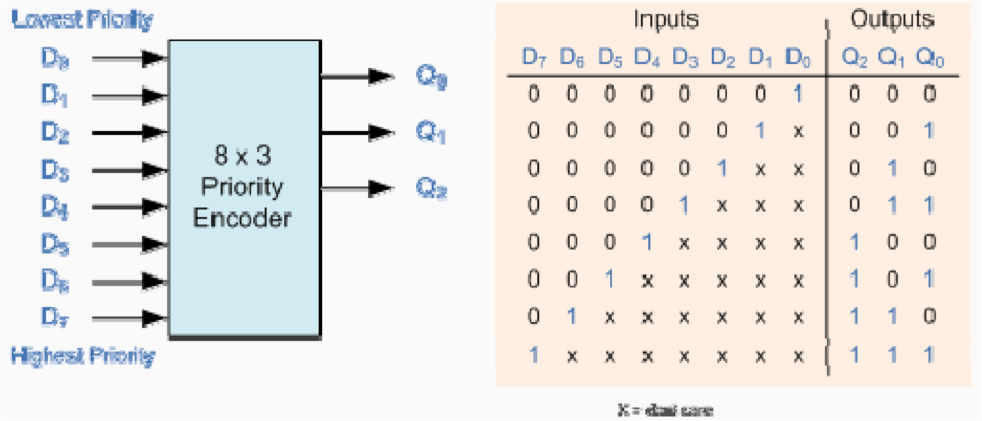
Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

One of the main disadvantages of standard encoders is that they can generate the wrong output code when there is more than one input present at logic level "1". For example, if we make inputs D_1 and D_2 HIGH at logic "1" at the same time, the resulting output is neither at "01" or at "10" but will be at "11" which is an output code that is different to the actual input present. One simple way to overcome this problem is to "Prioritize" the level of each input pin and if there was more than one input at logic level "1" the actual output code would only correspond to the input with the highest designated priority. Then this type of encoder are known as **Priority Encoders** or **P-encoder**.

Priority Encoders

Priority Encoders come in many forma and an example of an 8-input Priority Encoder along with its truth table is as shown below.

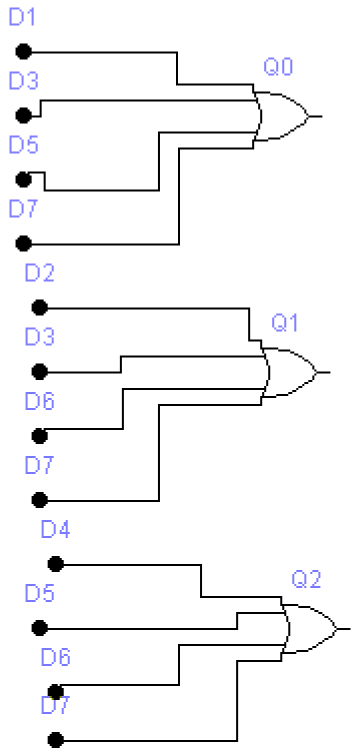
8-to-3 Bit Priority Encoder


Lowest Priority
D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
Highest Priority
8 x 3
Priority
Encoder
Q_0
Q_1
Q_2

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = don't care

Q0 = D1 + D3 + D5 + D7
Q1 = D2 + D3 + D6 + D7
Q2 = D4 + D5 + D6 + D7



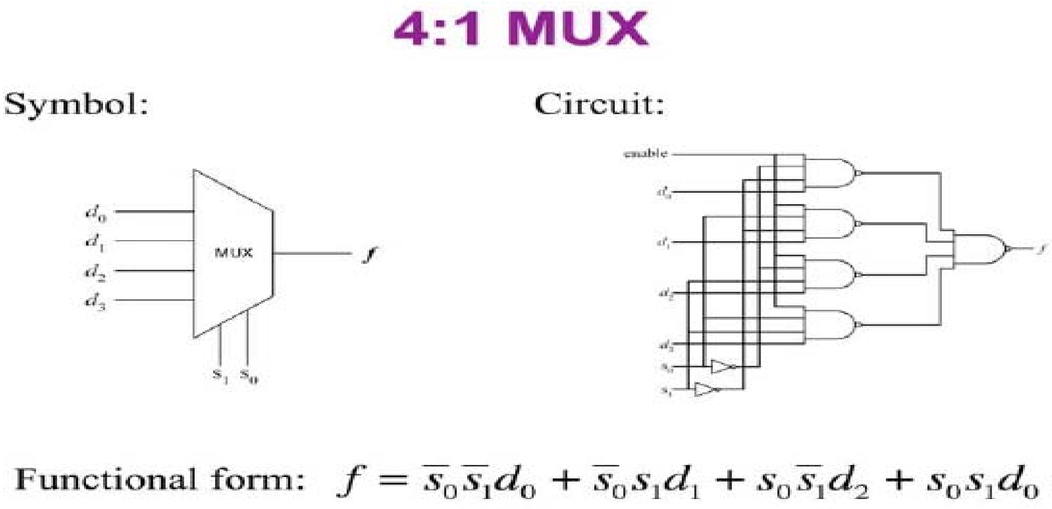
H.W)) Design Encoder BCD to Binary number .

Multiplexers & De-multiplexers

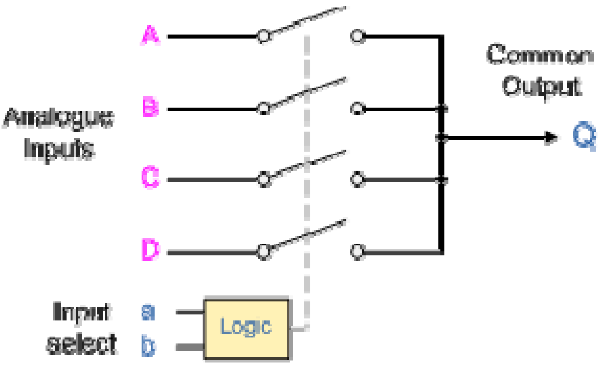
The Multiplexer

Multiplexers is a digital logic device that has 2^n data input lines and a single output, the logic input to n inputs select one of 2^n data inputs to be connected to the output. Sometimes is simply called "Mux" or "Muxes", that act like a very fast acting rotary switch. They connect multiple input lines 2, 4, 8, 16 etc one at a time to a common output line and are used as one method of reducing the number of logic gates required in a circuit. **Multiplexers** are individual Analogue Switches as opposed to the "mechanical" types such as normal conventional switches and relays. Selection of particular inputs is controlled by asset of selection lines.

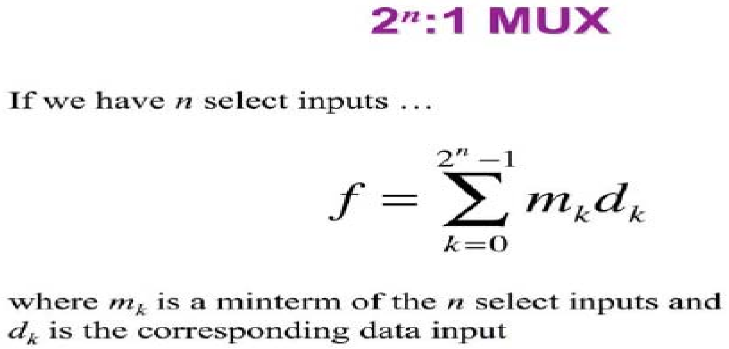
Normally, 2^n input lines requires n selection lines where bit. An example of a Multiplexer is shown below.



Functional form: $f = \overline{s}_0\overline{s}_1 d_0 + \overline{s}_0 s_1 d_1 + s_0 \overline{s}_1 d_2 + s_0 s_1 d_0$



F= m0 d0 + m1d1 + m2d2 + m3d3



2^n:1 MUX

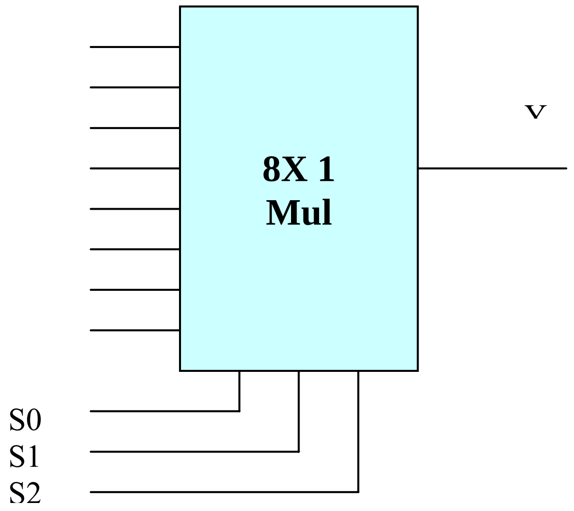
If we have n select inputs ...

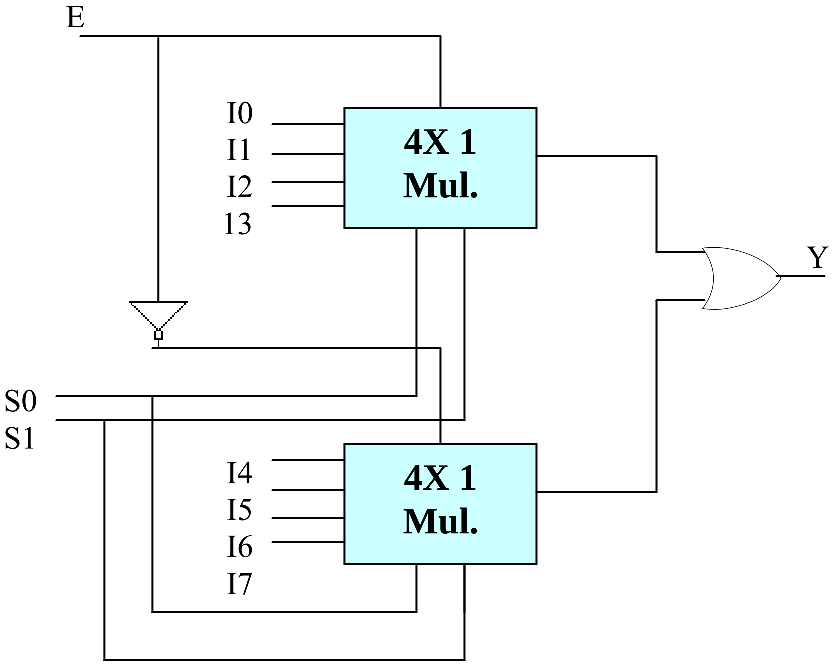
$$f = \sum_{k=0}^{2^n-1} m_k d_k$$

where m_k is a minterm of the n select inputs and d_k is the corresponding data input

Example: Construct (8X1) Mut. From 4X1 .
H.W)) Construct (16X1) Mut. From 4X1 9 use Decoder for selector).

Y	*S0*	*S1*	*E*
Y0	0	0	0
Y1	1	0	0
Y2	0	1	0
Y3	1	1	0
Y4	0	0	1
Y5	1	0	1
Y6	0	1	1
Y7	1	1	1

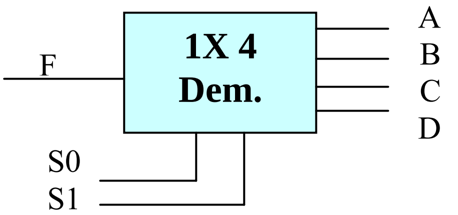


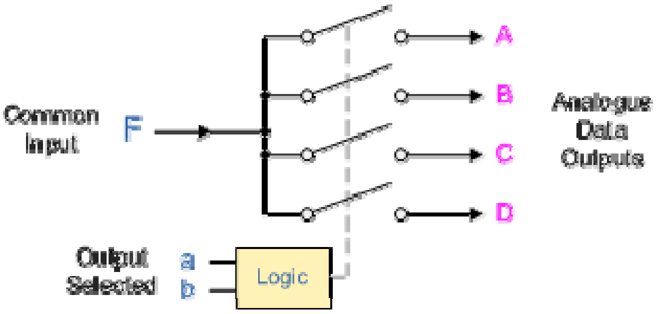


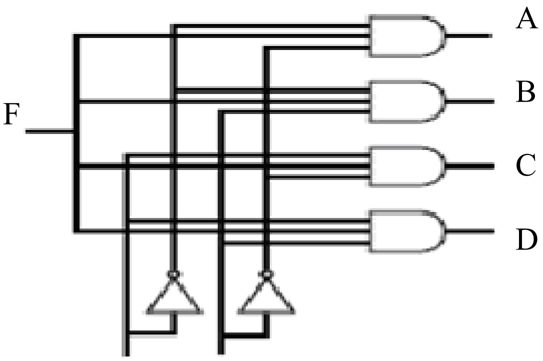
The De-multiplexer

De-multiplexers or "De-muxes", are the exact opposite of the *Multiplexers* . It has one single input data line and then switch it to any one of their individual multiple output lines one at a time. **The De-multiplexer** converts the serial data signal at the input to a parallel data at its output lines as shown below.

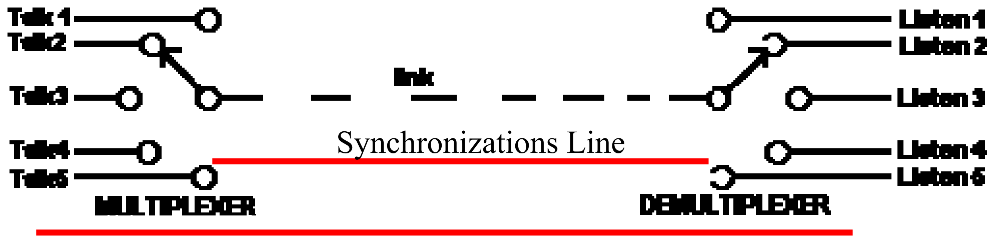
1-to-4 Channel De-multiplexer







Output Selected	Addressing	
	S0	S1
A	0	0
B	1	0
C	0	1
D	1	1



Muliplexing enables several signals to be sent over the same channel simultaneously.

In the top diagram, the Multiplexer rotary switch samples each channel in turn, and connects it to the link.

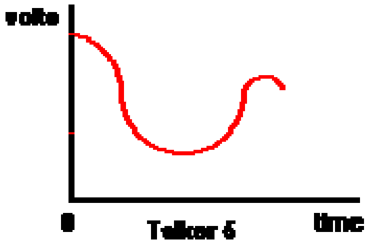
The Demultiplexer switch connects each listener in turn, to the link.

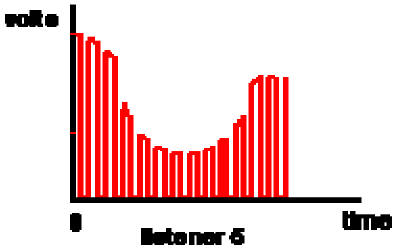
As long as the two switches are rotated in synchronism, Listener 1 will only hear Talker 1, etc.

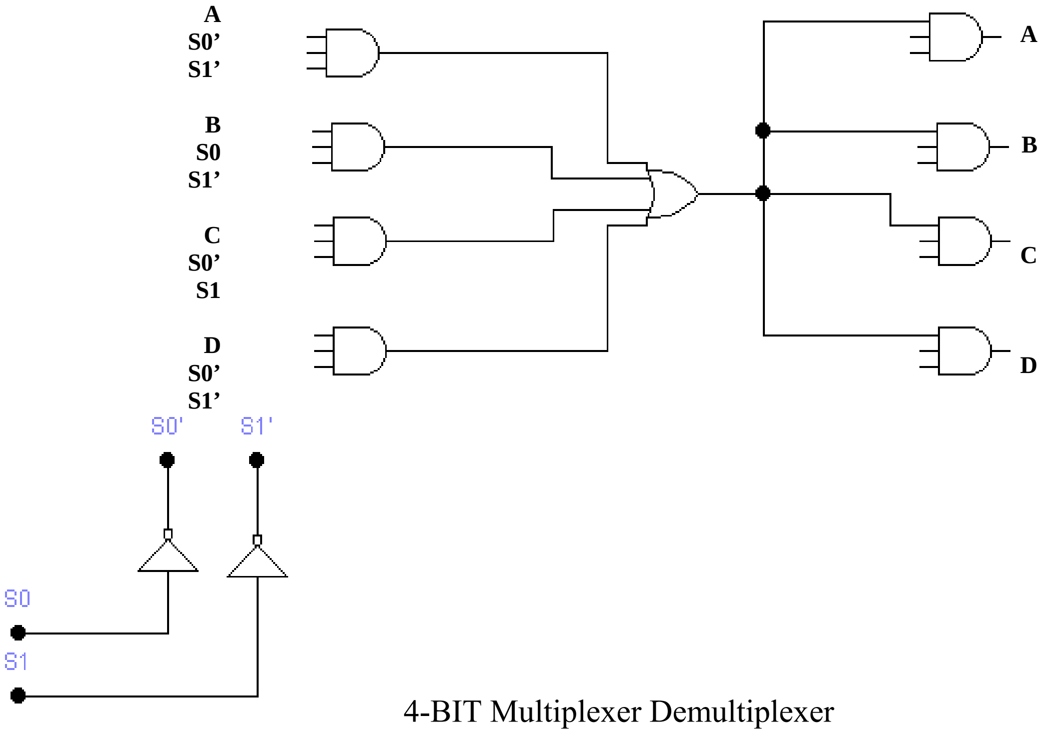
The minimum sample rate need only be twice the highest frequency of a talker signal, according to Nyquist.

In practice, electronic switches are used.

A synchronising signal is required to keep talkers and listeners in step.







4-BIT Multiplexer Demultiplexer