BROJNI SISTEMI, KONVERZIJE I LOGIČKA KOLA

Brojni sistemi

- DEKADNI – DEC S=10
  - cifre (0,1,2,3,4,5,6,7,8,9)
- HEKSADECIMALNI – HEX S=16

  cifre(0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)

  A=10 D=13

  B=11 E=14

  C=12 F=15
- OKTALNI – OCT S=8
  - cifre (0,1,2,3,4,5,6,7)
- BINARNI – BIN S=2
  - cifre (0,1)

Pozicioni brojni sistem

Svaka cifra ima zadatu težinu.

Opšti oblik:

$a_n a_{n-1} \ldots a_1 a_0 , a_{-1} a_{-2} \ldots a_{-m}$

tj.

$a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \ldots + a_1 \cdot b^1 + a_0 \cdot b^0 + a_{-1} \cdot b^{-1} + a_{-2} \cdot b^{-2} + \ldots + a_{-m} \cdot b^{-m}$

a – cifra

b – osnova (baza)

n+1 – broj celobrojnih cifara

m – broj decimala

Primer

Binarni	Oktalni	Decimalni	Heksadecimalni
00000	00	00	00
00001	01	01	01
00010	02	02	02
00011	03	03	03
00100	04	04	04
00101	05	05	05
00110	06	06	06
00111	07	07	07
01000	10	08	08
01001	11	09	09
01010	12	10	0A
01011	13	11	0B
01100	14	12	0C
01101	15	13	0D
01110	16	14	0E
01111	17	15	0F
10000	20	16	10
10001	21	17	11
10010	22	18	12

Primer

- Broj $701{,}625_{10}$ konvertovati u heksadecimalni brojni sistem.

  701 : 16 = 43 i ostatak 13 → D

  43 : 16 = 2 i ostatak 11 → B

  2 : 16 = 0 i ostatak 2

  ---

  rezultat: 2BD

Primer

- Broj $701{,}0625_{10}$ konvertovati u oktalni brojni sistem.

  701 : 8 = 87 i ostatak 5

  87 : 8 = 10 i ostatak 7

  10 : 8 = 1 i ostatak 2

  1 : 8 = 0 i ostatak 1

  ---

  rezultat: 1275

Konverzija iz binarnog u oktalni i heksadecimalni brojni sistem

- Drugačijim grupisanjem bitova:
  - po tri bita za oktalni brojni sistem
  - po četiri bita za heksadecimalni brojni sistem
- Primer:

  $01100111_2 = ?_8$

  $001\ 100\ 111_2 = 147_8$

- Primer:

  $01100111_2 = ?_{16}$

  $01100111_2 = 67_{16}$

heksadecimalni ←→ oktalni

- Preko binarnog brojnog sistema.
- Primer:

  $A3_{16} = 1010\ 0011_2$

  $0010\ 100\ 011_2 = 243_8$

AND

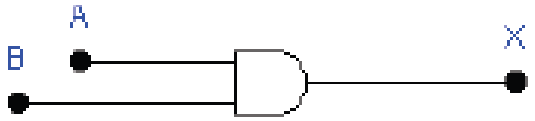
- Algebraically

  x = A.B

  or

  x = AB

- Pictorially



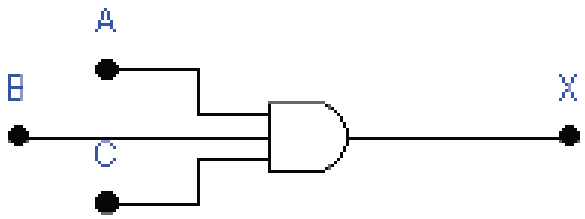
- Truth Table

A	B	x
0	0	0
0	1	0
1	0	0
1	1	1

- i.e. true if both inputs are true
- You can think of 1 (HIGH) as TRUE and 0 (LOW) as FALSE (because AND is just the familiar Boolean AND)

AND (there's more)

- AND can also be extended to more inputs
- x = A.B.C



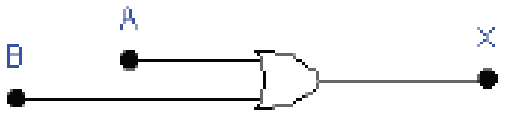
- **Truth Table**

A	B	C	x
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Only outputs a 1 if all inputs are 1

OR

- x = A + B

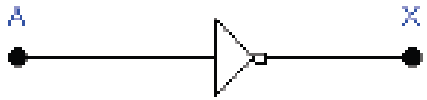


- **Truth table**

A	B	x
0	0	0
0	1	1
1	0	1
1	1	1

NOT

- **x = A'**
- **also written x = $\overline{A}$**

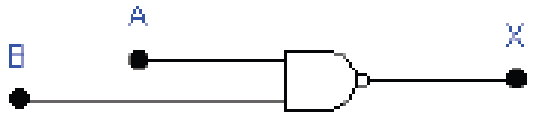


- Truth table

A	x
0	1
1	0

NAND (Not AND)

- x = (A.B) '



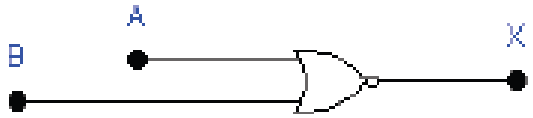
- Truth Table

A	B	x	[AND was]
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

- NAND is a *universal* gate

NOR (Not OR)

- x = (A + B) '



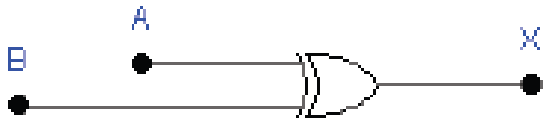
- Truth table

A	B	x	[OR was]
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

- NOR is a *universal* gate

XOR (Exclusive OR)

- **x = (A + B).(A.B) '**
  = A.B' + A'.B



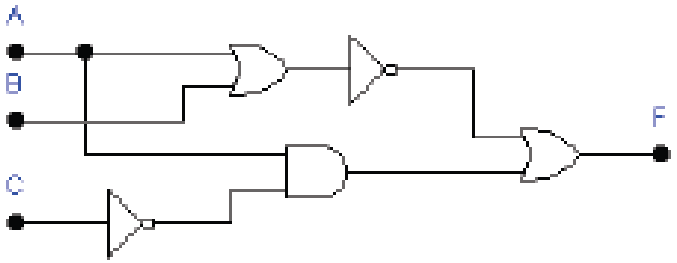
- Truth Table

A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

- either A or B but not both.

Example 1

- Start with a circuit



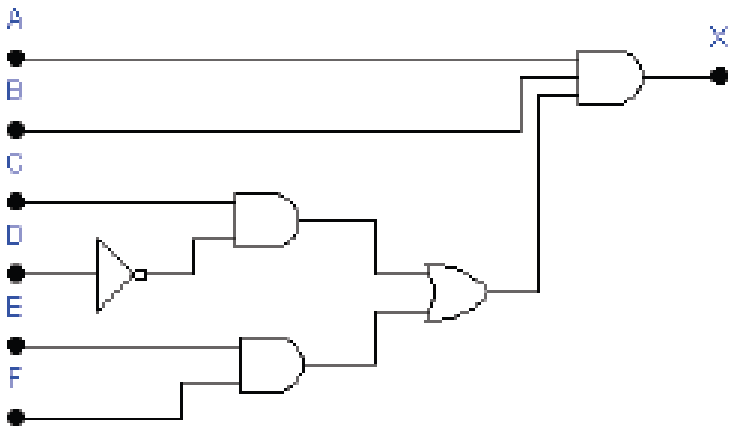
F = (A+B) ' + (A.C ')

A	B	C	(A+B) '	A.C'	F
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Example 2

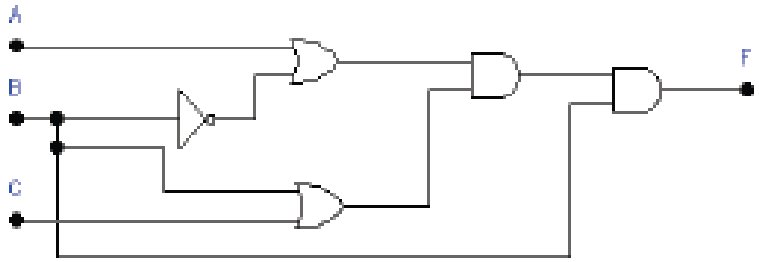
- Start with an expression

  X = A.B. (C.D ' + E.F)



Example

- F = B.((A+B').(B+C))



- Needs 2 AND gates, 2 OR gates, 1 NOT gate, therefore is expensive.

- ***But*** F is also A.B

A	B	C	A+B'	B+C	(A+B').(B+C)	F	A.B
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

- We can show this using Boolean algebra (later)