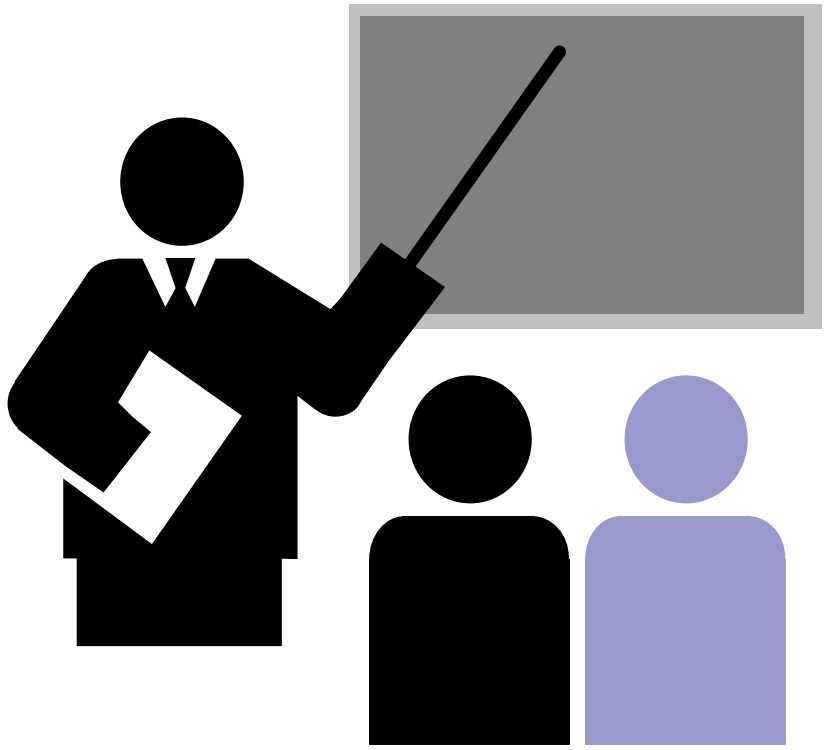
ITI 1100
Digital System I

Numeric systems

- Numeric Systems
- Binary Systems



Numeric System (base B)

- **Positional to Polynomial Notation**:

- $(N)_B = d_{n-1} B^{n-1} + d_{n-2} B^{n-2} + \ldots + d_1 B^1 + d_0 B^0 \cdot d_{-1} B^{-1} + \ldots + d_{-m} B^{-m}$

  Integral Part . Fractional *Part*

  - . = radix point
  - B = base or radix
  - d_i = value of the $i+1^{th}$ digit to the left of the point
  - d_{-j} = valeur du j^{th} digit to the the right of the point
  - n = number of integral digits in N
  - m = number of fractional digits in N

Most Known Numeric Systems

- B = 10 (Decimal)
  - digits: (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
  - $(34.25)_{10} = 3x10^{1} + 4x10^{0} + 2x10^{-1} + 5x10^{-2}$
- B = 2 (Binary)
  - digits: (0, 1)
  - $(101.01)_{2} = 1x2^{2} + 0x2^{1} + 1x2^{0} + 0x2^{-1} + 1x2^{-2}$

Most Known Numeric Systems

- B = 8 (Octal)
  - digits: (0, 1, 2, 3, 4, 5, 6, 7)
  - $(42.3)_8 = 4x8^1 + 2x8^0 + 3x8^{-1}$

- B = 16 (Hexadecimal)
  - digits: (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)
  - $(52.A6)_{16} = 5x16^1 + 2x16^0 + 10x16^{-1} + 6x16^{-2}$

Binary Order

n	2^n	n	2^n
0	1	8	256
1	2	9	512
2	4	10	1,024
3	8	11	2,048
4	16	12	4,096
5	32	13	8,192
6	64	14	16,384
7	128	15	32,768

Representation

Binary hexadecimal Decimal Octal

$2^3 2^2 2^1 2^0$	r=16	r=10	r=8
0 0 0 0	0	0	0
0 0 0 1	1	1	1
0 0 1 0	2	2	2
0 0 1 1	3	3	3
0 1 0 0	4	4	4
0 1 0 1	5	5	5
0 1 1 0	6	6	6
0 1 1 1	7	7	7
1 0 0 0	8	8	
1 0 0 1	9	9	
1 0 1 0	A		
1 0 1 1	B		
1 1 0 0	C		
1 1 0 1	D		
1 1 1 0	E		
1 1 1 1	F		

Conversion (I)

- How to move from a numeric representation to another one (change of base *B*)?

- Algorithme 1: Definition
  - favorable for conversions **towards** a decimal representation

$$(N_1)_A \rightarrow (N_2)_{10}$$

Conversion (II)

- Algorithme 2:
  Successives divisions and multiplications
  - favorable for conversions **from** a decimal system

$$(N_1)_{10} \rightarrow (N_2)_B$$

$$(34.625)_{10} \rightarrow (?)_2$$

Decimal to Binary

$$(34.625)_{10} \rightarrow (100010.101)_2$$

Integral Part:			Fractional Part:			
			↓			
34 ÷ 2 =	17	r **0**	0,625 x 2	=	**1**	,25
17 ÷ 2 =	8	r **1**	0,25 x 2	=	**0**	,5
8 ÷ 2 =	4	r **0**	0,5 x	=	**1**	
4 ÷ 2 =	2	r **0**				
2 ÷ 2 =	1	r **0**				
1 ÷ 2 =	0	r **1**				
	↑					

Binary ↔ Hexadecimal

- Binary → Hexadecimal (2^4)
  - Form groups of 4 bits starting from the point.
  - Each group of 4 bits represents an hexadecimal number.

$$(10110100.001011)_2 = (1011\ \ 0100.\ 0010\ \ 1100)_2$$
$$= (B\ \ 4\ .\ 2\ \ C)_{16}$$

Binary ↔ Hexadecimal

- Hexadecimal → Binary
  - Convert each hexadecimal number to its binary equivalent (4 bits).

$$261.35_{16} = (2\ \ 6\ \ 1\ .\ 3\ \ 5)_{16}$$
$$= (0010\ \ 0110\ \ 0001\ .\ 0011\ \ 0101)_2$$

Binary $\leftrightarrow$ Octal

- Binary $\rightarrow$ Octal
  - Form groups of 3 bits starting from the point.
  - Chaque groupe de 3 bits représente directement un chiffre octal.

$$(10110100.001011)_2 = (010\ 110\ 100\ .\ 001\ 011\ )_2$$
$$= (\ 2\ \ 6\ \ 4\ .\ \ 1\ \ \ \ 3\ )_8$$

Binary ↔ Octal

- Octal → Binary
  - Convert each octal number to its binary equivalent (3 bits).

$$(261.35)_8 = (2\ \ 6\ \ 1\ .\ 3\ \ 5)_8$$
$$= (010\ \ 110\ \ 001\ .\ 011\ \ 101)_2$$

The Basic Binary Addition

	0	0	1	1
	+0	+ 1	+0	+1
	--	--	--	--
sum	0	1	1	10

(sum of 0 and carryover of 1)

Examples:

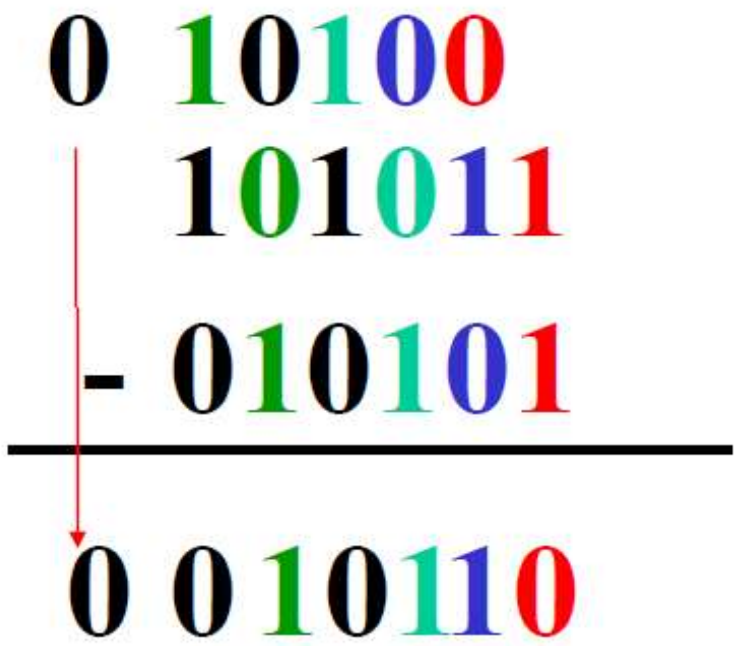
1 0 0 1	0 0 0 1	1 1 0 0
+ 0 1 1 0	+ 1 0 0 1	+ 0 1 0 1
1 1 1 1	1 0 1 0	1 0 0 0 1

carryover

The Basic Binary Subtraction

	0	0	1	1
	- 0	- 1	- 0	- 1
	--	--	--	--
difference	0	11	1	0

(Diff. of 1 and borrower of 1)



Complements in Digital Systems

- Complement are used to simplify the subtraction and for logical manipulation. There are 2 type of complements for each base:

Radix Complement (r's complement)

- r's CF(N) = $r^n - (N)_r$

  Where n is the number of digits in $(N)_r$

  2's CF $(101001)_2 = 2^6 - (101001)_2$

  $= (100000)_2 - (101001)_2$

  $= (010111)_2$

Radix-1 Complement (r-1's complement)

- r-1's CF(N) = $(r^n - 1)r - (N)_r$

  Where n is the number of digits in $(N)_r$

  1's CF $(1011000)_2$ =

  $(10000000 - 1)_2 - (1011000)_2$

  $= (0100111)_2$

1’s CF in Binary Systems

- 1’s Complement Form:
  - All bits are inverted by changing 1 to 0 and each 0 to 1.

Original	**One's Complement**
10011001	01100110
10000001	01111110
11110000	00001111
11111111	00000000
00000000	11111111

2's CF in Binary Systems

- 2's CF:
  - Represents both positive and negative values, it automatically includes the sign bit.
    - **2's Complement = 1's Complement + '1' (LSB)**

$$(10011001)_{2CF} = (10011001)_{1CF} + \text{'1'}$$

$$= \quad 01100110$$

$$+ \ 00000001$$

$$\text{----------}$$

$$(01100111)_2$$

Unsigned Binary Integers

- N is an integer represented in an n-bit word: $d_{n-1}d_{n-2}d_{n-3} \ldots d_2d_1d_0$
- In an unsigned representation, N is represented in terms of positional weighing: $0 \leq N \leq 2^n - 1$

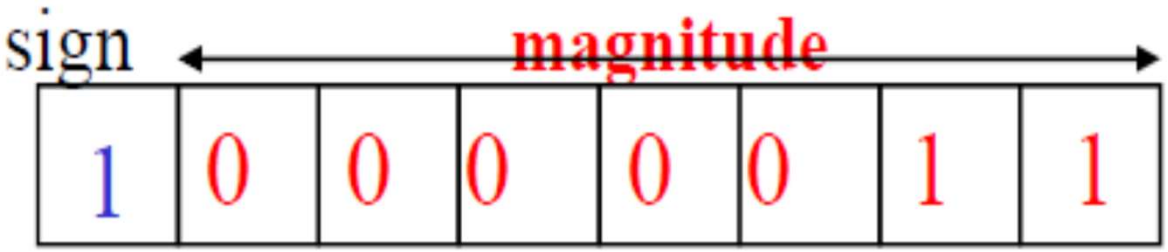
Signed Binary Integers

- +/-N n-bit word: $d_{n-1}d_{n-2}d_{n-3} \ldots d_2d_1d_0$ where:
  - d_{n-1} represents the **SIGN** (0 for positive and 1 for negative)
  - d_{n-1} represents **|N|** (absolute value)
  - $0 \leq |N| \leq 2^{n-1} - 1$

Signed Binary Integers

- +/-N is represented in 3 forms:
  - **Signed Magnitude Form** ($0 \leq |N| \leq 2^{n-1} - 1$)

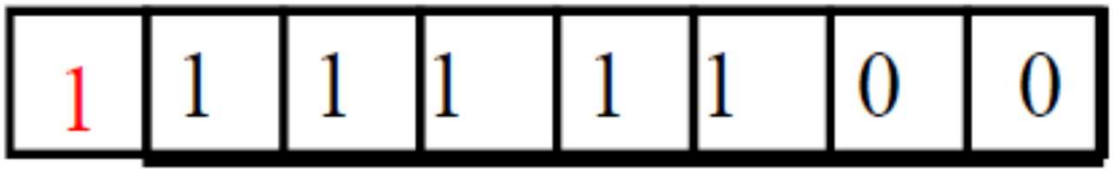
-3 = 10000011 = (-) 0000011



Signed Binary Integers

- +/-N is represented in 3 forms:
  - **1's Complement Form (1's CF)**

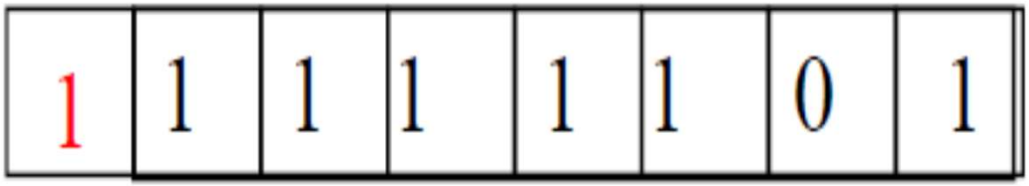
    -3 = 10000011 = (-) 1111100



Signed Binary Integers

- +/-N is represented in 3 forms:
  - **2's Complement Form (2's CF)**

    -3 = 10000011 = (-) 1111101



Comparison

4-bit word	UBI	SMF	1's CF	2's CF
0000	0	+0	+0	0
0001	1	+1	+1	+1
0010	2	+2	+2	+2
0011	3	+3	+3	+3
0100	4	+4	+4	+4
0101	5	+5	+5	+5
0110	6	+6	+6	+6
0111	7	+7	+7	+7
1000	8	−0	−7	−8
1001	9	−1	−6	−7
1010	10	−2	−5	−6
1011	11	−3	−4	−5
1100	12	−4	−3	−4
1101	13	−5	−2	−3
1110	14	−6	−1	−2
1111	15	−7	−0	−1

Comparison

- In both SMF and 1'S CF there 7 positive and negative integers, 2 0's. In 2's CF there also 7 positive and negative integers and one 0.
- For an n-bit wide word, for both SMF and 1's CF have 2^{n-1} - 1 positive and negative numbers and 2 0's while for 2's CF, there are 2^{n-1} - 1 positive and 2^n – 1 negative numbers and one).

Subtraction with 2's CF

- 2's CF is used to convert substraction into addition (only adders are used).
- From the minus sign properties of the 2's CF.
  - **A + (-A) = 0** **A + 2'sCF(A) = 0**
  - **- (-A) = A** **2'sCF(2'sCF(A)) = A**
  - **Thus:**
    - **A – B = A + (-B)** ➔
      - **A – B = A + 2'sCF(B)**

Substraction with 2's CF

- Discard the carry over, both the subtrahend and result are in 2's CF.
- Example: A = 1010100 B = 1000011

A – B = A = (-B) = A + 2's(B)

2's(1000011) = (0111100 + '1') = 0111101

A – B = 1010100 – 1010100 = 0010001

```
          1010100
    +     0111101
    ---------------
  1 0010001
```

Discard end carry

Subtraction with 1's CF

- Both the subtrahend and result are in 1's CF.
- If the result of the addition has a carry over, add it to the last bit, otherwise take the 1's CF of the result and it is negative.
- Example: 110101 – 100101:

  110101

  + 011010 (1's CF)

(carry over) 1001111 → 001111 + 000001 = 010000

The result is positive so it stays at it is.

Overflow

- A specific size/length register can only handle a specific **range** of numbers.d the carry out.
  - A 4-bit wide register can handle a range of positive integers from 0 – 15
  - For an 8-bit wide: 0 – 255
  - That is for an n-bit wide: 0 – $(2^n – 1)$

Overflow

- Overflow occurs in the addition when the sign of the operands are the same and if the sign bit of the result is not the same as the sign bits of the operands.
  - The addition of 2 positive numbers results in a negative number.
  - The addition of 2 negative numbers results in a positive number

Binary Codes for Characters

- Weighted Codes.
  - Are positionally weighted.
  - The *Binary Coded Decimal* (BCD)
    - Represents digits 0 - 9
    - 4 bits are used.
    - Weights: 2^3, 2^2, 2^1, 2^0 or 8421 code
    - Ex: $(9750)_{10} = (1001\ 0111\ 0101\ 000)_{BCD}$
- Non-Weighted Codes.
  - Are NOT positionally weighted.
  - The EXCESS-3 CODE: XS3

BCD and Excess-3 Codes

Decimal	BCD	Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
...		
7	0111	1010
8	1000	1011
9	1001	1100

Even and Odd Parity Codes

DECIMAL DIGIT	8421	EVEN PARITY	ODD PARITY
0	0000	00000	00001
1	0001	00011	00010
2	0010	00101	00100
3	0011	00110	00111
4	0100	01001	01000
5	0101	01010	01011
6	0110	01100	01101
7	0111	01111	01110
8	1000	10001	10000
9	1001	10010	10011

Binary Codes for Characters

- Alphanumeric Codes.
  - ASCII (7 bits)
  - ISO Latin 9 (8 bits)
  - Unicode
  - UTF-8
  - etc.

ASCII Code (1)

	$B_7B_6B_5$							
$B_4B_3B_2B_1$	000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB		7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

ASCII Code (2)

Control Characters:

NULL	NULL	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
Sl	Shift in	US	Unit separator
SP	Space	DEL	Delete

ASCII Representation of ‘Digital’

Character	Binary Code	Hexadecimal Code
D	1000100	44
i	1101001	69
g	1100111	67
i	1101001	69
t	1110100	74
a	1100001	61
l	1101100	6C

Boolean Algebra

- Set of two elements: 0 and 1 (True/False)
- Set of basic logic operators:

  {AND, OR, NOT}
- Boolean or logic variables: Any valid letter or word (A, b, x, y, car, etc.) whose values can only be 1 or 0 (True/false).
- Used to optimize logic functions.

Truth Tables and Logic Functions

- Any logic function defined by *n* logic variables logiques results to *True* or *False* (0 or 1).
- A Truth Table represents one or several functions of n variables in terms of its 2^n inputcombinations.
- Any logic function can be defined by its truth table and represented by logic gates.

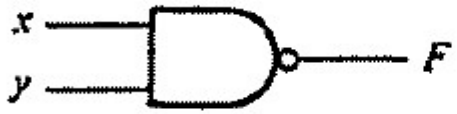
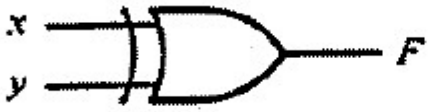
Truth Table: Addition of 3 Binary Digits

x	y	CarryIn	Sum	CarryOut
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

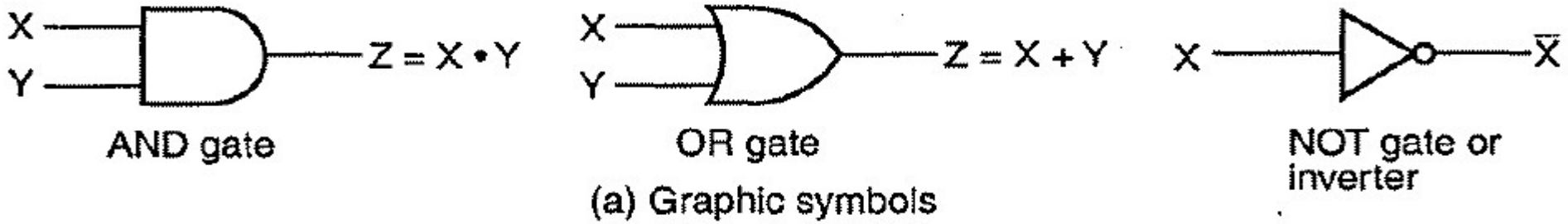
Basic Logic Gates (1)

Name	Graphic symbol	Algebraic function	Truth table
AND		$F = xy$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1

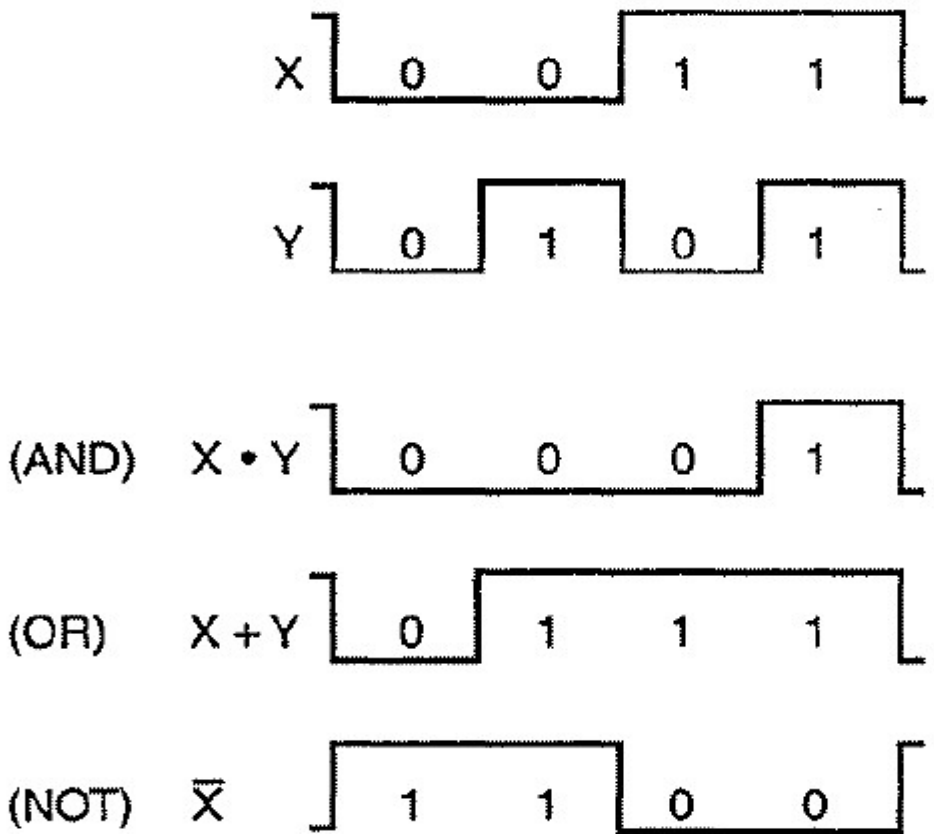
Basic Logic Gates (2)

Name	Graphic symbol	Algebraic function	Truth table
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= x \odot y$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Basic Logic Functions (1)

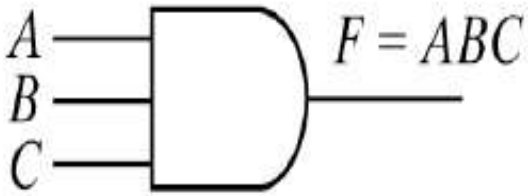


(a) Graphic symbols

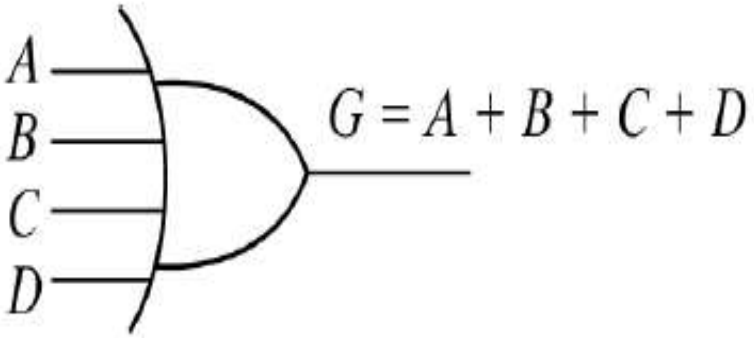


Gates with Multiple Inputs

Gates may have more than 2 input signals

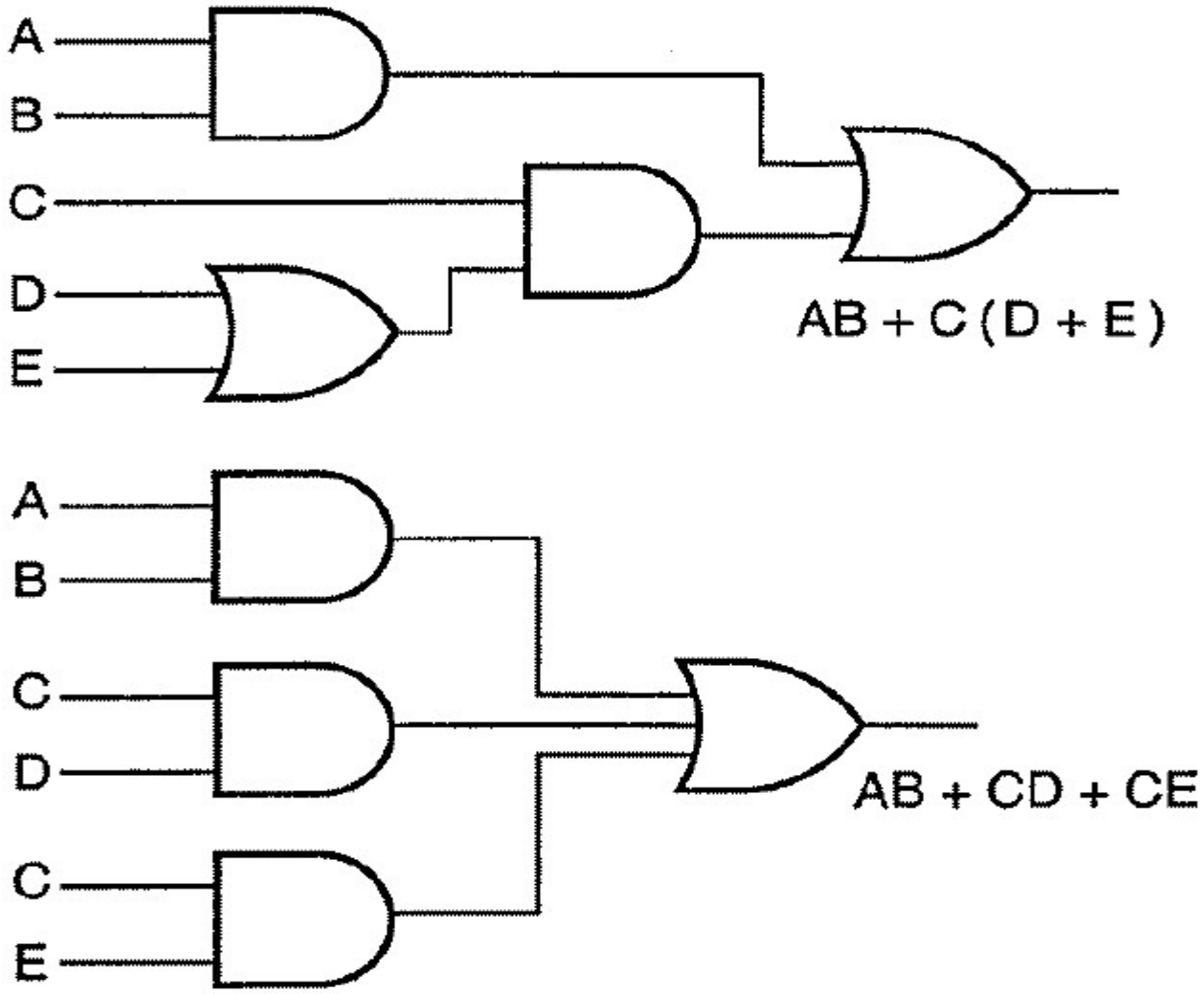


(a) Three-input AND gate



(b) Four-input OR gate

Circuit Examples



Boolean Identities (1)

$$x + 0 = x$$
$$x \cdot 1 = x$$
$$x + x' = 1$$
$$x \cdot x' = 0$$
$$x + x = x$$
$$x \cdot x = x$$
$$x + 1 = 1$$
$$x \cdot 0 = 0$$
$$(x')' = x$$

Basic Boolean Algebra Theorems

Boolean Identities (2)

$x + y = y + x$	Commutative
$xy = yx$	Commutative
$x + (y + z) = (x + y) + z$	Associative
$x(yz) = (xy)z$	Associative
$x(y + z) = xy + xz$	Distributive
$x + yz = (x+y)(x+z)$	Associative

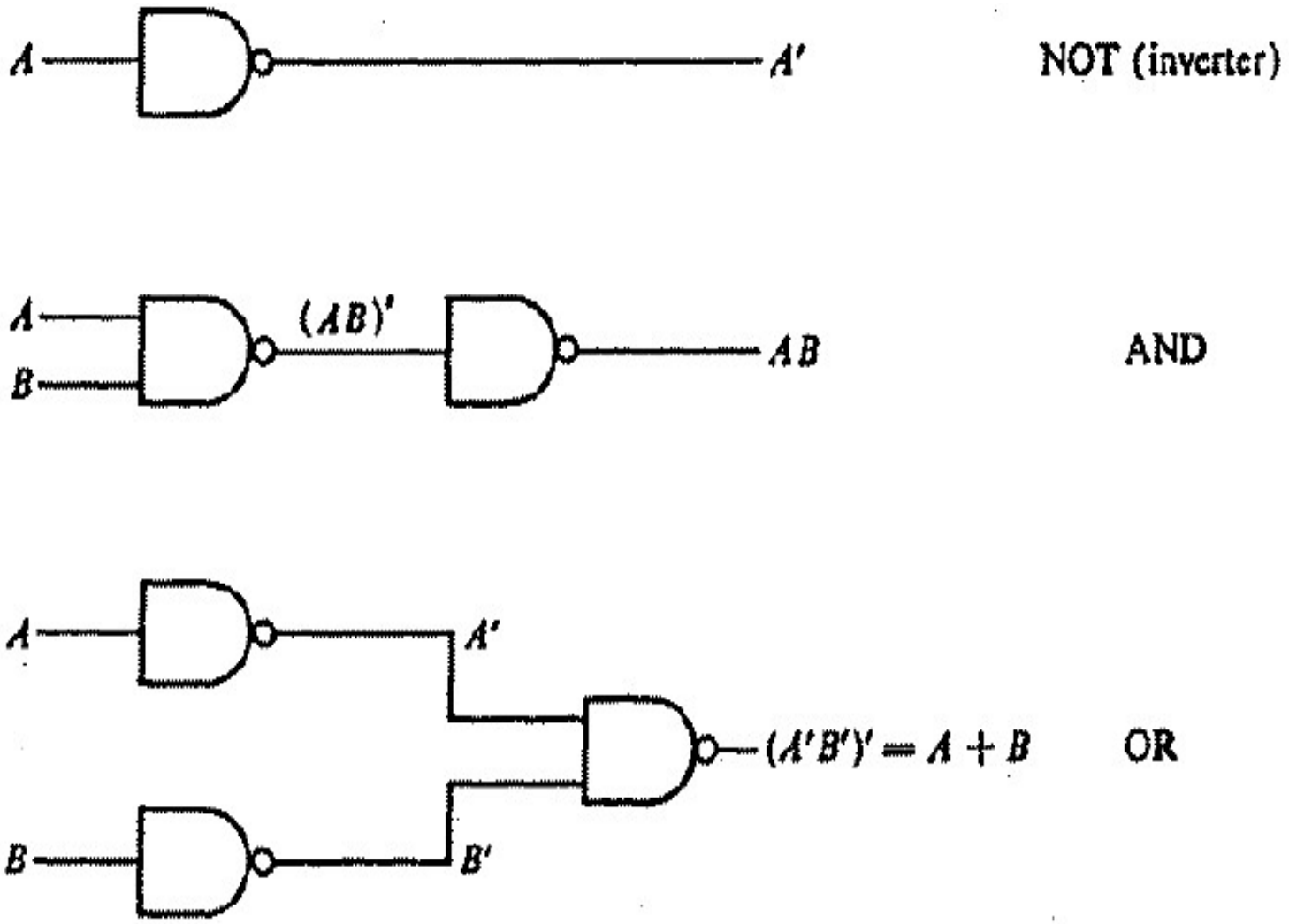
Boolean Identities (3)

$(x + y)' = x' y'$	DeMorgan
$(xy)' = x' + y'$	DeMorgan
$x + xy = x$	Absorption
$x(x + y) = x$	Absorption

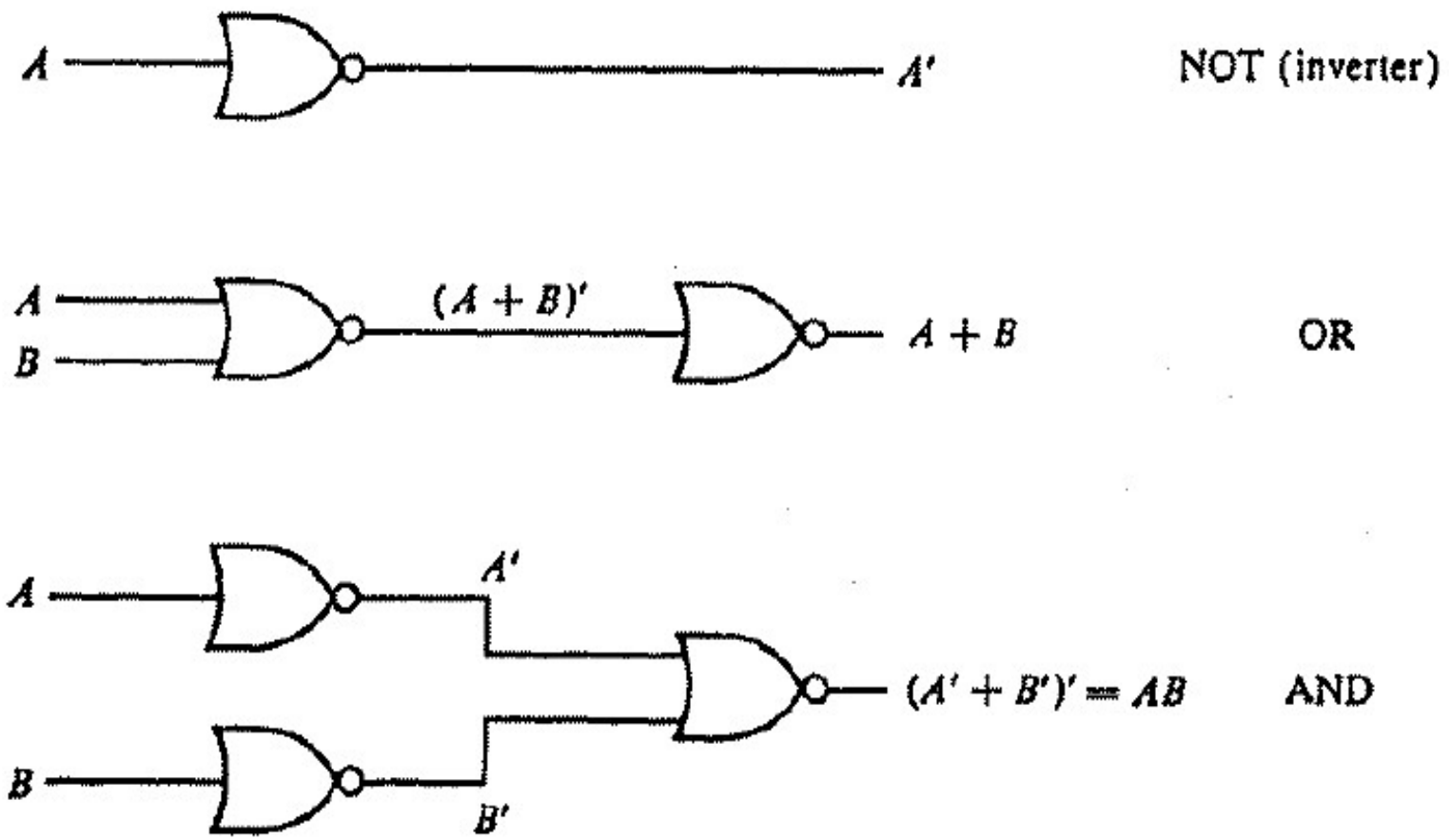
The NAND and NOR Gates

- Functionally complete.
- Known as Universal Logic gates.
- Can be used to design any of the basic logic gates (*AND, OR, NOT*) as well as any other design.
- Faster and cheaper to produce.

The NAND Gate



The NOR Gate



Simplification with Boolean Algebra

F(ABC) = A'BC' + A'BC + AB'C' + AB'C' + ABC + ABC'

= A'B(C' + C) + AB'(C + C') + AB(C + C')

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

= A'B + AB' + AB

= A'B + A(B' + B)

= A'B + A(1)

= A + A'B

Simplification with Boolean Algebra

F(ABC) = AC' + A'B + A'B'C' = ?

= AC' + **A'B(1 + C')** + A'B'C'

= AC' + **A'B +** A'BC' + A'B'C'

= AC' + A'B + A'C' (B + B')

= AC' + A'B + A'C'

= AC' + A'B + A'C'

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

= A'B + C'