Digital Systems and Information Part II

Overview

- Arithmetic Operations
  - General Remarks
  - Unsigned and Signed Binary Operations
- Number representation using Decimal Codes
  - BCD code and Seven-Segment Code
- Text representation
  - Alphanumeric Codes - ASCII and Unicode
- Sound and Speech representation
  - Can we talk digitally?
- Image and Video representation
  - Can we see digitally?

Arithmetic Operations

- Arithmetic operations with numbers in base ***r*** follow the same rules as for decimal numbers
  - Examples: addition, subtraction, and multiplication in base-2

Carries:		**111**
Augend:		10110
Addend:	+	01110
Sum:		100100

Borrows:				**11**
Minuend:		00110	-	11101
Subtrahend:	-	11101		00110
Difference:				-10111

Multiplicand:		1010
Multiplier:	x	101
		1010
	+	**0000**
		1010
Product:		110010

- In Digital Computers arithmetic operations are done with the binary number system (base-2) - Binary Arithmetic
- Binary subtraction is done by binary addition! Why?
  - It is much more simple to do it that way
  - Simple building block, called adder, can be implemented
  - One Adder block can be used for both binary addition and subtraction

Unsigned Binary Subtraction

- Binary subtraction done by 2's complement addition
  - Assume two **n**-bit unsigned numbers **M** and **N**, **M - N** can be done as follows:
  - Take the 2's complement of **N** and add it to **M** ==> **M** + (2^n - **N**)
  - The sum is M + (2^n - N) = M – N + 2^n
  - If M ≥ N, the sum produces an end carry, 2^n. We can discard it, leaving the correct result M – N.
  - If M < N, the sum does not produces an end carry because it is equal to 2^n – (N - M), which is the 2's complement of N – M. To obtain the correct result take the 2's complement of the sum, i.e., 2^n – (2^n – (N - M)) = (N - M) and place a minus sign in front.
- Examples:

15	1111	(15)
- 5	+ 1011	(2's compl. of 5)
10	1 1010	(the sum)

1 → discard carry

5	0101	(5)
- 15	+ 0001	(2's compl. of 15)
-10	0110	(the sum)

no carry

correction is needed

-1010 ("-" 2's compl. of sum)

- What about binary subtraction by 1's complement addition?
  - I leave this for you as a home work (see the course web page) !!!

Signed Binary Addition

- Assume two **n**-bit signed numbers **M** and **N** in signed-2's complement format
- **M + N** is done as follows:
  - Add **M** and **N** including their sign bits
  - A carry out of the sign bit position is discarded
- Obtained sum is always correct!
  - the sum is in signed-2's complement format
- Examples:

```
   (- 9)   1|0111        (- 9)    1|0111        (+ 9)   0|1001        (+ 9)    0|1001
 + (+ 5)   0|0101      + (- 5)    1|1011      + (+ 5)   0|0101      + (- 5)    1|1011
   -------------         --------------         -------------         --------------
   (- 4)   1|1100        (-14)   11|0010        (+14)   0|1110        (+ 4)   10|0100
                                 └► discard                                   └► discard
```

Signed Binary Subtraction

- Assume two **n**-bit signed numbers **M** and **N** in signed-2's complement format
- **M - N** is done by addition as follows:
  - Take the 2's complement of **N** (including the sign bit) and add it to **M** (including the sign bit)
  - A carry out of the sign bit position is discarded
- Obtained result is always correct!
  - the result is in signed-2's complement format
- Examples:

$$\begin{array}{rrr} & (-\ 9) & 1|0111 \\ - & \underline{(-\ 5)} & \underline{1|1011} \\ & (-\ 4) & \end{array} \longrightarrow \begin{array}{rr} & 1|0111 \\ + & \underline{0|0101} \\ & 1|1100 \end{array} \qquad \begin{array}{rrr} & (-\ 9) & 1|0111 \\ - & \underline{(+\ 5)} & \underline{0|0101} \\ & (-14) & \end{array} \longrightarrow \begin{array}{rr} & 1|0111 \\ + & \underline{1|1011} \\ & 11|0010 \end{array}$$

↳ discard

Binary Floating-point Operations

- Assume two binary floating-point numbers $F1 = M1 \times 2^{E1}$ and $F2 = M2 \times 2^{E2}$
- Multiplication: $F = F1 \times F2 = \mathbf{M} \times 2^{\mathbf{E}}$ (how to find **M** and **E**)
  - $F = F1 \times F2 = (M1 \times 2^{E1}) \times (M2 \times 2^{E2}) = \mathbf{(M1 \times M2)} \times 2^{\mathbf{(E1 + E2)}}$
- Division: $F = F1 / F2 = \mathbf{M} \times 2^{\mathbf{E}}$ (how to find **M** and **E**)
  - $F = F1 / F2 = (M1 \times 2^{E1}) / (M2 \times 2^{E2}) = \mathbf{(M1 / M2)} \times 2^{\mathbf{(E1 - E2)}}$
- Addition: $F = F1 + F2 = \mathbf{M} \times 2^{\mathbf{E}}$ (how to find **M** and **E**)
  - If $E1 \geq E2$ then $F = F1 + F2 = (M1 \times 2^{E1}) + (M2 \times 2^{E2}) =$
    $= M1 \times 2^{E1} + (M2 \times 2^{(E2-E1)}) \times 2^{E1} = \mathbf{(M1 + (M2 \times 2^{-(E1-E2)}))} \times 2^{\mathbf{E1}}$
- Subtraction: $F = F1 - F2 = \mathbf{M} \times 2^{\mathbf{E}}$ (how to find **M** and **E**)
  - If $E1 \geq E2$ then $F = F1 - F2 = (M1 \times 2^{E1}) - (M2 \times 2^{E2}) =$
    $= M1 \times 2^{E1} - (M2 \times 2^{(E2-E1)}) \times 2^{E1} = \mathbf{(M1 - (M2 \times 2^{-(E1-E2)}))} \times 2^{\mathbf{E1}}$
- After each operation, M has to be *normalized* (if necessary) by shifting it to the left and decrementing E until a nonzero bit appears in the first position.
- Example:

$$\begin{array}{r} 5.00 \\ -\ 2.75 \\ \hline 2.25 \end{array} \qquad \begin{array}{r} (+0.101000)_2 \times 2^{(011)_2} \\ -\ (+0.101100)_2 \times 2^{(010)_2} \\ \hline \end{array} \longrightarrow \begin{array}{r} (+0.101000)_2 \times 2^{(011)_2} \\ -\ (+0.010110)_2 \times 2^{(011)_2} \\ \hline \mathbf{(+0.010010)_2} \times 2^{\mathbf{(011)_2}} \end{array} \longrightarrow \begin{array}{c} \textit{normalized result} \\ (+0.100100)_2 \times 2^{(010)_2} \end{array}$$

Number Representation using Decimal Codes

- The binary number system is used in digital computers
- BUT people are accustomed to the decimal system
- We can resolve this difference by
  - converting decimal numbers to binary
  - performing all arithmetic calculations in binary
  - converting the binary result back to decimal
- You already know how to do this!
- Digital computers can do this as well, BUT:
  - We have to store the decimal numbers in the computer in a way that they can be converted to binary
  - Since the computer can accept only 1’s and 0’s, we must represent the decimal digits by a code that contains 1’s and 0’s

Binary Coded Decimals (BCD) (1)

- BCD code is the most commonly used code
- Each decimal digit is coded by a 4-bit string called BCD digit
- A decimal number is converted to a BCD number by replacing each decimal digit with the corresponding BCD digit code
- Example:

$$(369)_{10} = (\underbrace{0011}_{3}\ \underbrace{0110}_{6}\ \underbrace{1001}_{9})_{BCD} = (101110001)_2$$

- A BCD number needs more bits than its equivalent binary value!
- However, the advantages of using BCD are:
  - BCD numbers are decimal numbers even though they are represented 1s and 0s
  - Computer input/output data are handled by people who use the decimal system
  - Computers can store decimal numbers using BCD, convert the BCD numbers to binary, perform binary operations, and convert the result back to BCD

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

Note: the binary combinations **1010** through **1111** are not used and have no meaning in the BCD code

Binary Coded Decimals (BCD) (2)

- Converting a BCD number to a binary number

$(25)_{10} = \mathbf{(0010\ 0101)_{BCD}} = (0010)_2 \times 10^1 + (0101)_2 \times 10^0 =$

$= (0010)_2 \times (1010)_2 + (0101)_2 \times (0001)_2 =$

$= (10100)+(0101) = \mathbf{(11001)_2}$

- Converting a binary number to a BCD number

Convert the number $\mathbf{(11001)_2}$ by dividing it to $(1010)_2 = (10)_{10}$

$(11001)_2 / (1010)_2 = (0010)_2$ and Remainder = $(0101)_2$ Least significant BCD digit

$(0010)_2 / (1010)_2 = (0000)_2$ and Remainder = $(0010)_2$ Most significant BCD digit

$\mathbf{(11001)_2 = (\ 0010\ 0101\ )_{BCD} = (25)_{10}}$

- BCD Arithmetic
  - Digital computers can perform arithmetic operations directly with decimal numbers stored in BCD code
  - How is this done? (study the text book or go to internet for information)

Other Useful Decimal Codes: Excess-3 Code

- Given a decimal digit *n*, its corresponding excess-3 codeword is $(n+3)_2$
  - Example:
    n=5 → n+3=8 → $1000_{excess\text{-}3}$
    n=0 → n+3=3 → $0011_{excess\text{-}3}$

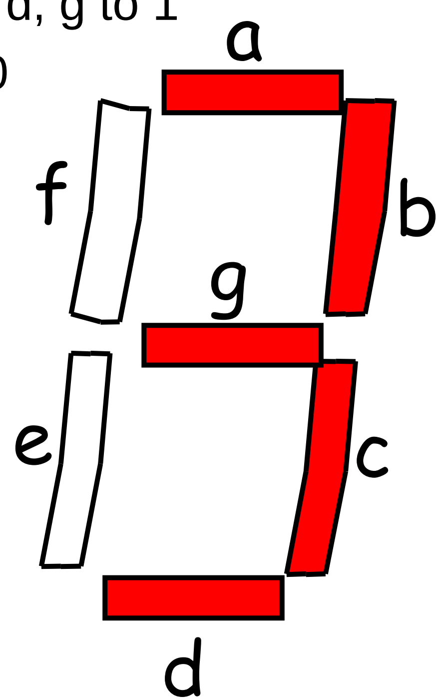
Decimal Digit	Excess-3 Digit
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

- Decimal number in Excess-3 code.
  - Example:
    $(158)_{10} = (\underset{1+3}{\underline{\mathbf{0100}}}\ \underset{5+3}{\underline{\mathbf{1000}}}\ \underset{8+3}{\underline{\mathbf{1011}}})_{excess\text{-}3} = (10011110)_2$

- Useful in some cases for digital arithmetic, e.g., decimal subtraction.

Another Useful Decimal Code: Seven-Segment Code

- Used to display numbers on seven-segment displays
- Seven-segment display:
  - 7 LEDs (light emitting diodes), each one controlled by an input
  - 1 means “on”, 0 means “off”
  - Display digit “3”?
    - Set a, b, c, d, g to 1
    - Set e, f to 0



Decimal Digit	7- Segment Code						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Text Representation using Alphanumeric Codes

- Digital computers need to handle data consisting not only of numbers, but also of letters
  - Alphanumeric character set of English includes:
    - The 10 decimal digits
    - The 26 letters of the alphabet (uppercase and lowercase letters)
    - Several special characters (more than three)
  - We need to code these symbols
    - The code must be binary – computers can handle only 0's and 1's
    - We need binary code of at least seven bits (2^7 = 128 symbols)
- American Standard Code for Information Interchange (ASCII)
  - 7-bit standard code for representing symbols of the English language
- Unicode
  - 16-bit standard code for representing the symbols of other languages

ASCII Code Table

American Standard Code for Information Interchange (ASCII)

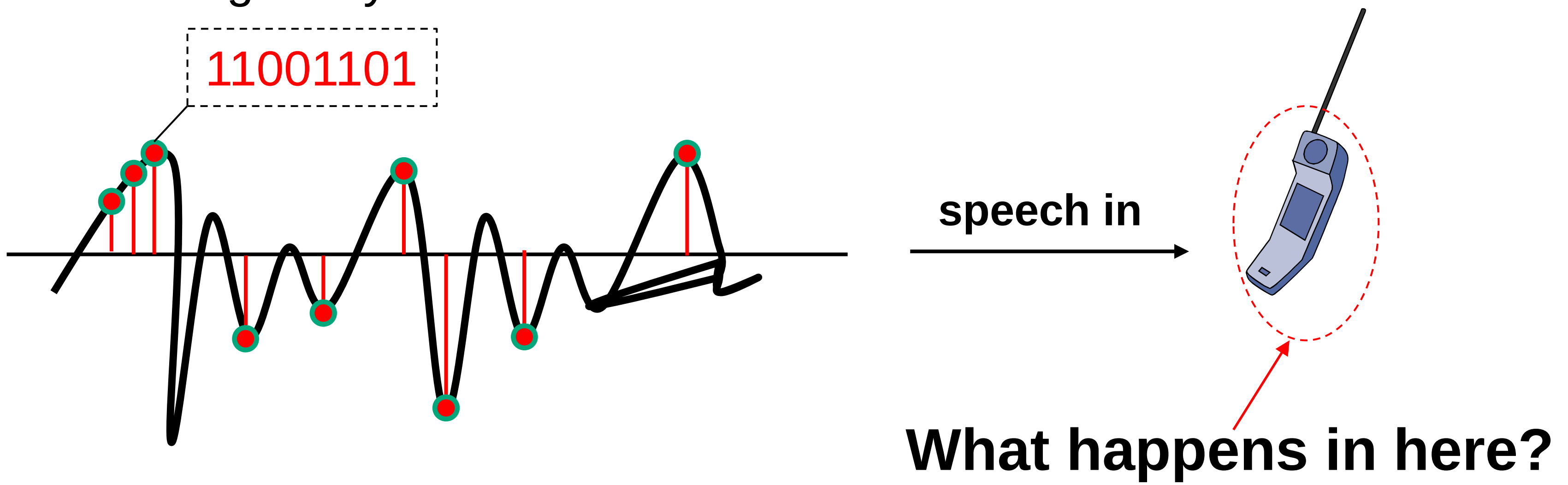
	$B_7B_6B_5$							
$B_4B_3B_2B_1$	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	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

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
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

How can we talk digitally?

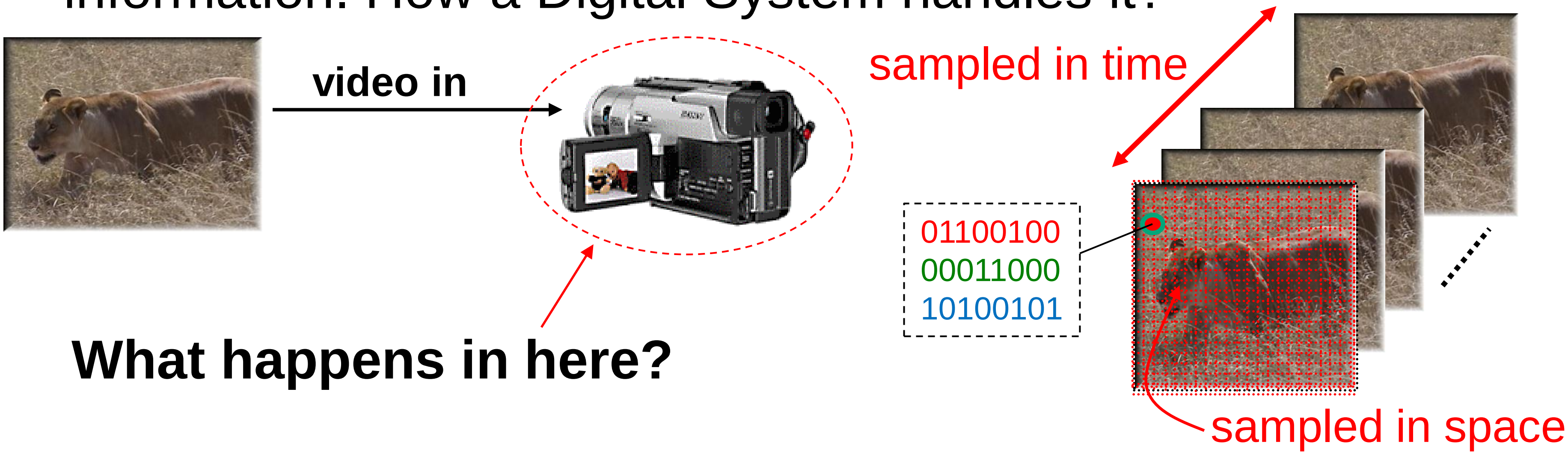
- Digital Systems manipulate discrete quantities of information
- Speech and music are continuous (non-discrete) quantities of information
- How a Digital System can handle this continuous information?



1) Signal is *sampled in time* at 8000 samples per second
2) Each sample is *quantized* and coded by a single byte
3) After (1) and (2) we get discrete quantity of information:
   - The cost is 64 Kbit/sec which is way too much!
   - Digital Signal Processing techniques allow us to bring this amount down to as low as 2.4 Kbit/s.

How can we see digitally?

- Image and video are continuous (non-discrete) quantities of information. How a Digital System handles it?



1) Video Signal is *sampled in time* at 24 images (frames) per second
2) Each image is *sampled in space* at 3.2 mega pixels per image
3) Each pixel is *quantized* and coded by 3 bytes (Red, Green, Blue)
4) After (1), (2), and (3) we get discrete quantity of information:
   - The cost is 1.8432 Gbit/sec which is huge!!!
   - Image Compression techniques (JPEG, MPEG-4, H.264) allow us to bring this amount down to several Mbit/s

Combinational Logic Circuits
Part I -Theoretical Foundations

Overview

- What is a combinational logic circuit?
- Boolean Algebra
  - Basic Logic Operations
  - Basic Identities
  - Basic Principles, Properties, and Theorems
- Boolean Function and Representations
- Truth Table
- Canonical and Standard Forms
- Karnaugh Maps (K-Maps)

Combinational Logic Circuits

- Digital Systems are made out of digital circuits
- Digital circuits are hardware components that manipulate binary information
- Certain well defined basic (small) digital circuits are called Logic Gates
- Gates have inputs and outputs and perform specific mathematical logic operations
- Outputs of gates are connected to inputs of other gates to form a digital combinational logic circuit.

Boolean Algebra

- To analyze and design digital combinational logic circuits we need a mathematical system
- A system called Boolean Algebra is used
- George Boole (1815-1864): “An investigation of the laws of thought” – a book published in 1854 introducing the mathematical theory of logic
- Boolean Algebra deals with binary variables that take 2 discrete values (0 and 1), and with logic operations
- Binary/logic variables are typically represented as letters: A,B,C,…,X,Y,Z or a,b,c,…,x,y,z
- Three basic logic operations:
  - **AND**, **OR**, **NOT** (complementation)

Basic Logic Operations

- **AND** operation is represented by operators "•" or "$\wedge$" or by the absence of an operator.
  - $Z = X \cdot Y$ or $Z = X \wedge Y$, or $Z = XY$ is read "Z is equal to X AND Y" meaning that:
  - $Z = 1$ if and only if $X = 1$ and $Y = 1$; otherwise $Z = 0$.
  - **AND** resembles binary multiplication:

$$0 \cdot 0 = 0, \qquad 0 \cdot 1 = 0,$$
$$1 \cdot 0 = 0, \qquad 1 \cdot 1 = 1$$

- **OR** operation is represented by operators "+" or "$\vee$".
  - $Z = X+Y$ or $Z = X \vee Y$ is read "Z is equal to X OR Y" meaning that:
  - $Z = 1$ if $X = 1$ or $Y = 1$, or if both $X = 1$ and $Y = 1$, i.e., $Z = 0$ if and only if $X = 0$ and $Y = 0$.
  - **OR** resembles binary addition, except in one case:

$$0 + 0 = 0, \qquad 0 + 1 = 1,$$
$$1 + 0 = 1, \qquad \mathbf{1 + 1 = 1}\ (\neq 10_2)$$

- **NOT** operation is represented by operator " ' " or by a bar over a variable.
  - $Z = X'$ or $Z = \overline{X}$ is read "Z is equal to NOT X" meaning that:
  - $Z = 1$ if $X = 0$; but $Z = 0$ if $X = 1$
  - **NOT** operation is also referred to as *complement* operation.

Basic Identities of Boolean Algebra

Let X be a boolean variable and 0,1 constants

1. $X + 0 = X$ -- Zero Axiom
2. $X \cdot 1 = X$ -- Unit Axiom
3. $X + 1 = 1$ -- Unit Property
4. $X \cdot 0 = 0$ -- Zero Property

5. $X + X = X$ -- Idempotence
6. $X \cdot X = X$ -- Idempotence
7. $X + X' = 1$ -- Complement
8. $X \cdot X' = 0$ -- Complement
9. $(X')' = X$ -- Involution

Boolean Algebra Properties

Let X,Y, and Z be boolean variables

- Commutative
  - 10. $X + Y = Y + X$
  - 11. $X \cdot Y = Y \cdot X$
- Associative
  - 12. $X + (Y+Z) = (X+Y) + Z$
  - 13. $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
- Distributive
  - 14. $X \cdot (Y+Z) = X \cdot Y + X \cdot Z$
  - 15. $X+(Y \cdot Z) = (X+Y) \cdot (X+Z)$
- DeMorgan's Theorem
  - 16. $(X + Y)' = X' \cdot Y'$
  - 17. $(X \cdot Y)' = X' + Y'$
- In general for DeMorgan,
  - $(X1 + X2 + \ldots + Xn)' = X1' \cdot X2' \cdot \ldots \cdot Xn'$,
  - $(X1 \cdot X2 \cdot \ldots \cdot Xn)' = X1' + X2' + \ldots + Xn'$

The Duality Principle

- The dual of an expression is obtained by exchanging (• and +), and (1 and 0) in it, provided that the precedence of operations is not changed
- Cannot exchange x with x'
- Example:
  - Find the dual of expression: $x'yz' + x'y'z$
  - Answer: $(x' + y + z') \cdot (x' + y' + z)$
- The dual expression does not always equal the original expression
- If a Boolean equation/equality is valid, its dual is also valid

The Duality Principle (cont.)

With respect to duality, Identities 1 – 8 and Properties 10 – 17 have the following relationship:

1. $X + 0 = X$	2. $X \bullet 1 = X$	(dual of 1)
3. $X + 1 = 1$	4. $X \bullet 0 = 0$	(dual of 3)
5. $X + X = X$	6. $X \bullet X = X$	(dual of 5)
7. $X + X' = 1$	8. $X \bullet X' = 0$	(dual of 7)

10. $X + Y = Y + X$	11. $X \bullet Y = Y \bullet X$	(dual of 10)
12. $X + (Y+Z) = (X+Y) + Z$	13. $X \bullet (Y \bullet Z) = (X \bullet Y) \bullet Z$	(dual of 12)
14. $X \bullet (Y+Z) = X \bullet Y + X \bullet Z$	15. $X+(Y \bullet Z) = (X+Y) \bullet (X+Z)$	(dual of14)
16. $(X + Y)' = X' \bullet Y'$	17. $(X \bullet Y)' = X' + Y'$	(dual of16)

Absorption Property (Covering)

- $X + X \bullet Y = X$ -- (absorption property)
- $X \bullet (X+Y) = X$ -- (dual absorption property)
- **Proof:**

  $$\begin{aligned} X + X \bullet Y &= X \bullet 1 + X \bullet Y \\ &= X \bullet (1 + Y) \\ &= X \bullet 1 \\ &= X \end{aligned}$$
- Can you prove the dual absorption property?

Consensus Theorem

- $XY + X'Z + YZ = XY + X'Z$ -- (theorem)
- $(X+Y)\bullet(X'+Z)\bullet(Y+Z) = (X+Y)\bullet(X'+Z)$ -- (dual theorem)
- **Proof:**

$$
\begin{aligned}
XY + X'Z + YZ &= XY + X'Z + (X+X')YZ \\
&= XY + X'Z + XYZ + X'YZ \\
&= (XY + XYZ) + (X'Z + X'ZY) \\
&= XY + X'Z
\end{aligned}
$$

- Can you prove the dual consensus theorem?

Boolean Function

- F(vars) = *Boolean expression*

set of binary variables

- Operators (+, •, ’)
- Variables
- Constants (0, 1)
- Groupings (parenthesis)

- Example: F(a,b) = a’• b + b’

  G(x,y,z) = x • (y + z’)

- Terminology:
  - *Literal:* A variable or its complement (Example: x or b’).
  - *Product term:* literals connected by “•” (Example: a’• b).
  - *Sum term:* literals connected by “+” (Example y + z’).

Boolean Function Representations

- Truth Table (unique representation)
- Boolean Equation
  - Canonical Sum-Of-Products (CSOP) form (unique)
  - Canonical Product-Of-Sums (CPOS) form (unique)
  - Standard Forms (NOT unique representations)
- Map (unique representation)

- We can convert one representation of a Boolean function into another in a systematic way
- Why do we need all these representations?

Truth Table

- Tabular form that *uniquely* represents the relationship between the input variables of a Boolean function and its output
  - Enumerates all possible combinations of 1's and 0's that can be assigned to binary variables
  - Shows binary value of the function for each possible binary combination
- Example:

F(x,y,z)

x	y	z	F
0	**0**	**0**	**1**
0	**0**	**1**	**0**
0	**1**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**0**	**1**
1	**0**	**1**	**1**
1	**1**	**0**	**0**
1	**1**	**1**	**0**

Truth Table (cont.)

- Assume a Boolean function $F(x_1,x_2,\ldots,x_{N-1},x_N)$ that depends on ***N*** variables
- Question1: How many columns are there in the truth table of $F(x_1,x_2,\ldots,x_{N-1},x_N)$?
- Question2: How many rows are there in the truth table of $F(x_1,x_2,\ldots,x_{N-1},x_N)$?
- Answer Q1: columns = **N + 1**
  - a column is needed for each variable and 1 column is needed for the values of the function
- Answer Q2: rows = $\mathbf{2^N}$
  - there are 2^N possible binary combinations for *N* variables

Truth Table (cont.)

- Truth table: a unique representation of a Boolean function
- If two functions have identical truth tables, the functions are equivalent (and vice-versa)
- Truth tables can be used to prove equality theorems
  - Proof of the DeMorgan's Theorem: **(X + Y)' = X' • Y'**

X	Y	X + Y	F1 = (X + Y)'
0	**0**	**0**	**1**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**1**	**0**

X	Y	X'	Y'	F2 = X' • Y'
0	**0**	**1**	**1**	**1**
0	**1**	**1**	**0**	**0**
1	**0**	**0**	**1**	**0**
1	**1**	**0**	**0**	**0**

Observe: F1 and F2 have identical truth tables => F1 = F2, i.e., the theorem is proved

- The size of a truth table grows exponentially with the number of variables involved
- This motivates the use of other representations!