Topics

Module 1 : Fundamentals of Digital Systems and logic families Digital signals, digital circuits, AND, OR, NOT, NAND, NOR and Exclusive-OR operations, Boolean algebra, examples of IC gates, number systems-binary, signed binary, octal hexadecimal number, binary arithmetic, one's and two's complements arithmetic, codes, error detecting and correcting codes, characteristics of digital ICs, digital logic families, TTL, Schottky TTL and CMOS logic, interfacing CMOS and TTL, Tri-state logic.

Introduction

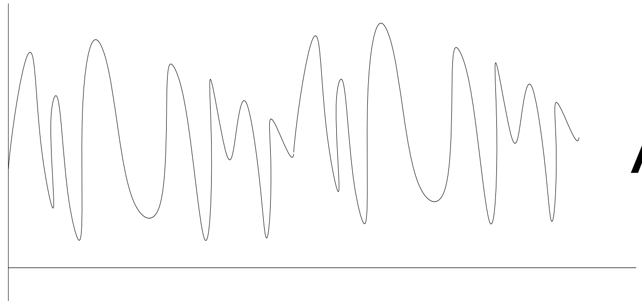
- We will start by explaining the name of our course, “Digital Logic” starting by:
  - Digital
  - Then
  - Logic

Digital Systems I

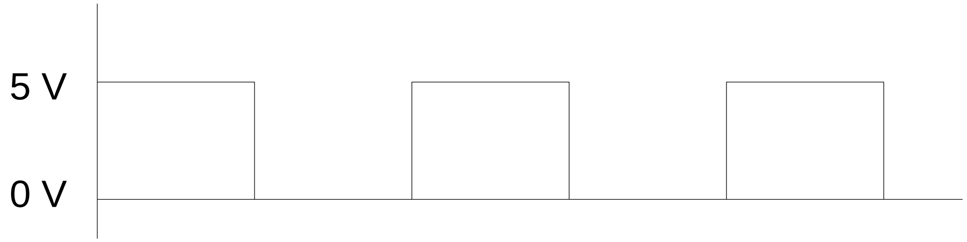
- Differentiate between two types of systems or signals:
  - Analog: found in nature and initially all systems were analog.
  - Digital: evolved from analog systems as a result of technology development to avoid the disadvantages of the first class.

Digital Systems II

- Examples:
- Analog signal: speech.



- Digital signal: clock pulses for a computer.



- Analog system: traditional audio phones.
- Digital system: computers.

Digital Systems III

- Digital or discrete values can be:
  - Binary: one bit only, either 0 or 1.
  - Binary code: a series or sequence of bits to represent specific values.
- From hardware point of view , i.e. electronically, these values are represented by electrical signals (voltage or current).

  0 → Low → 0 volt.

  1 → High → 5 volt.
- The applications or the device that we deal with can produce digital signals directly, e.g. computers, or produce analog signals, e.g. phones.
- We can convert an analog signal to digital using Analog to Digital Converter (ADC) and for the reverse operation we use a Digital to Analog Converter (DAC).
- Examples: voice over IP (VoIP) applications.

Big Picture

E.g. Computers — Digital System

↑ Interconnected

E.g. Memory — Digital Modules

↑ Combined/Interconnected

E.g. Flip Flops, AND gates, etc. — Digital Circuits or Logic Circuits

- Digital circuit: called logic circuits since they process data represented as binary signals via logic gates or components (specifies the relation between inputs logically).

Logic

- *Logic:* is something related to human's way of thinking where hypothesis and assumptions are made to simplify things.
- In our situation, we use the binary symbols 0, 1 or we can equate them with other values according to the needs of the moment.
- Example:

0 = false = No = 0 volt or 1 volt ...

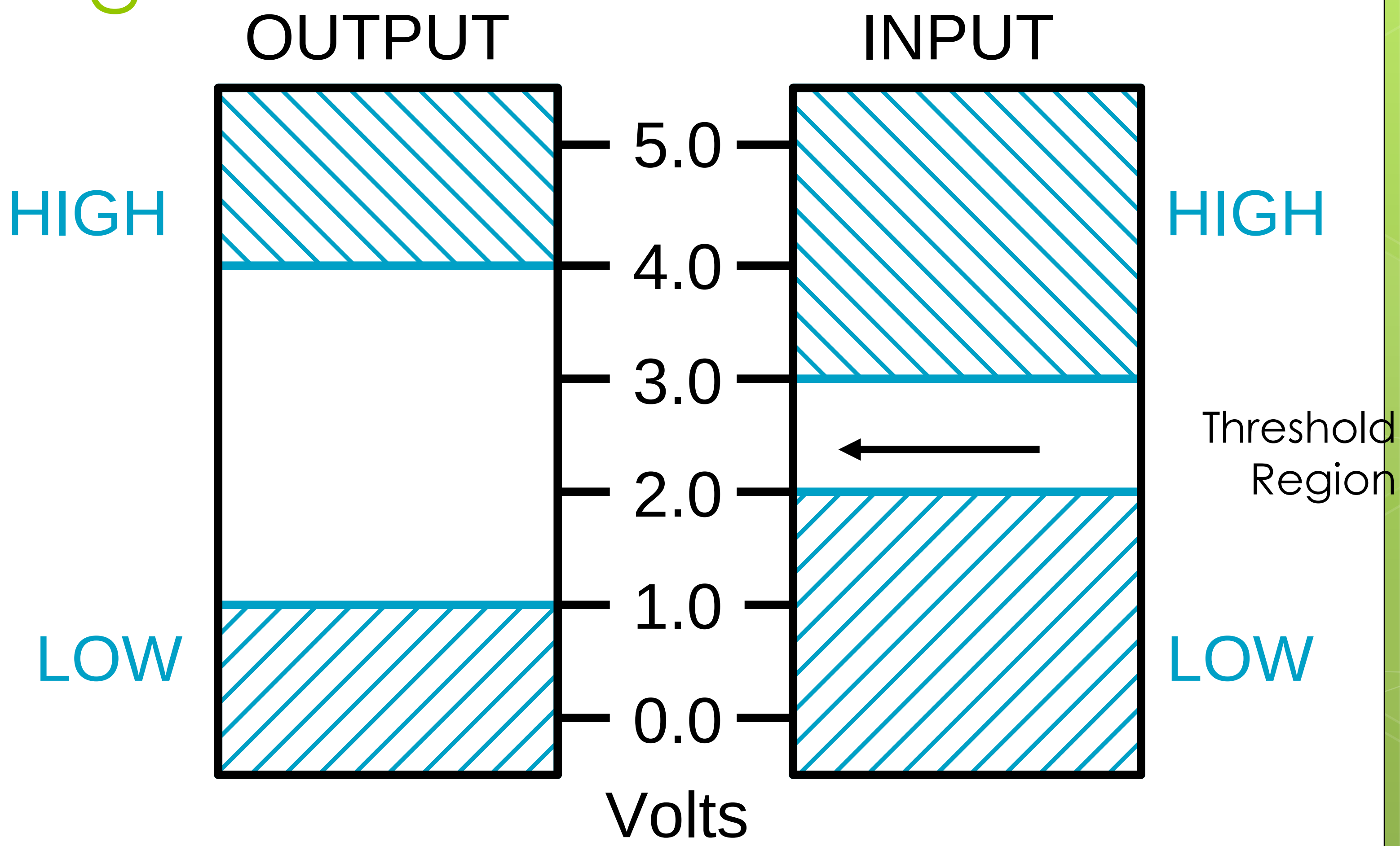
1 = true = Yes = 5 volt or 12 volt ...

- Also, in logic we have only two states or values, either up or down, nothing between.
- The logic that deals with states between 0 and 1 (i.e. more than two states) is called fuzzy logic.
- Although it seems limited but it proves to be very useful. Example: computers are based on the two states operation and they are very powerful.

Signal

- An information variable represented by physical quantity.
- For digital systems, the variable takes on discrete values.
- Two level, or binary values are the most prevalent values in digital systems.
- Binary values are represented abstractly by:
  - digits 0 and 1
  - words (symbols) False (F) and True (T)
  - words (symbols) Low (L) and High (H)
  - and words On and Off.
- Binary values are represented by values or ranges of values of physical quantities

Signal Example – Physical Quantity: Voltage



Binary Values: Other Physical Quantities

- **What are other physical quantities represent 0 and 1?**
  - **CPU** Voltage
  - **Disk** Magnetic Field Direction
  - **CD** Surface Pits/Light
  - **Dynamic RAM** Electrical Charge

ANALOG GOES DIGITAL

- Photography
- Video
- Audio
- Automobile applications
- Telephony/Telecommunications
- Traffic lights
- Special effects

ADVANTAGES OF DIGITAL PROCESSING

- Reproducibility of results
- Ease of design
- Programmability
- Speed
- Noise tolerance

1.1 Digital Systems (1-2)

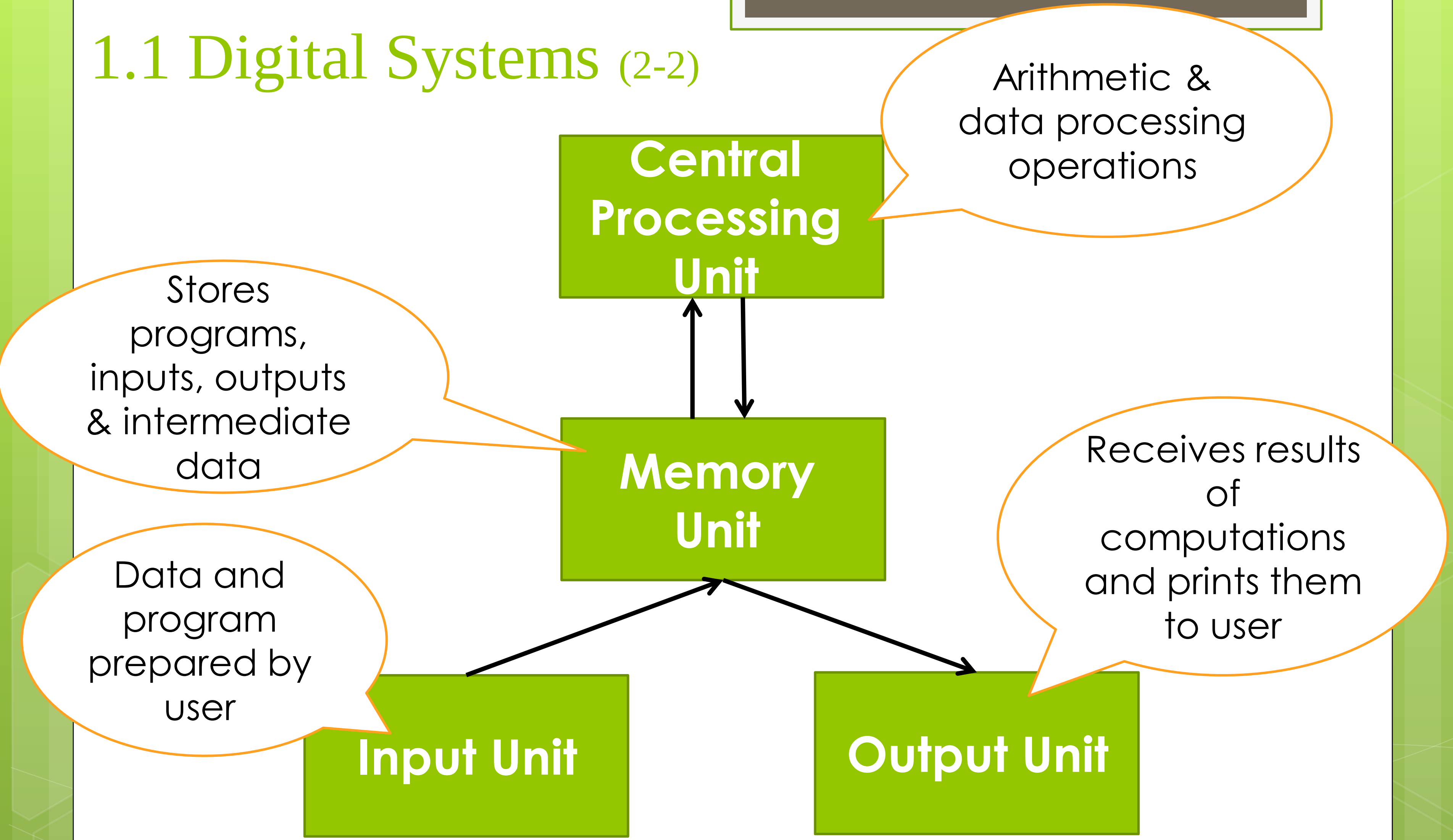
- **Digital** system is a system that uses ***discrete*** values such as numbers and letters.

- The signal in most digital systems use two values :
  0 and 1 which called a **bit**.

- Discrete elements of information are represented with a group of bits called **binary codes.**

- **Thus**, **Digital** system is a system that manipulates ***discrete*** elements of information represented internally in binary form.

1.1 Digital Systems (2-2)



Block diagram of a digital Computer

Logic Circuits

- A collection of individual logic gates connect with each other and produce a logic design known as a Logic Circuit
- The following are the types of logic circuits:
  - Decision making
  - Memory
- A gate has two or more binary inputs and single output.

Basic Logic Gates

- The following are the three basic gates:
  - NOT
  - AND
  - OR
- Each logic gate performs a different logic function. You can derive logical function or any Boolean or logic expression by combining these three gates.

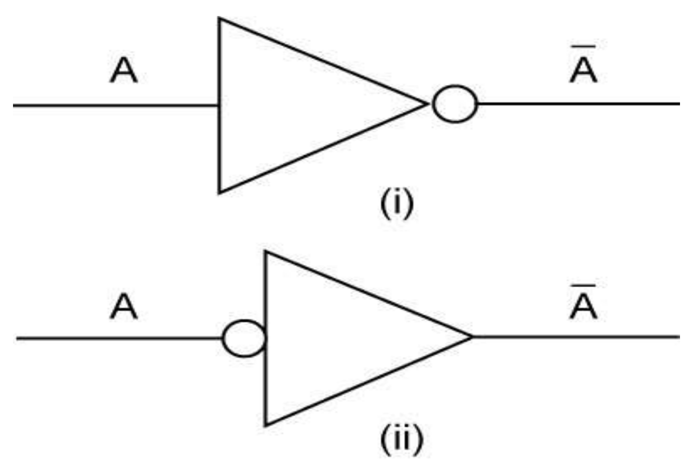
NOT Gate

- The simplest form of a digital logic circuit is the inverter or the NOT gate
- It consists of one input and one output and the input can only be binary numbers namely; 0 and 1

the truth table for

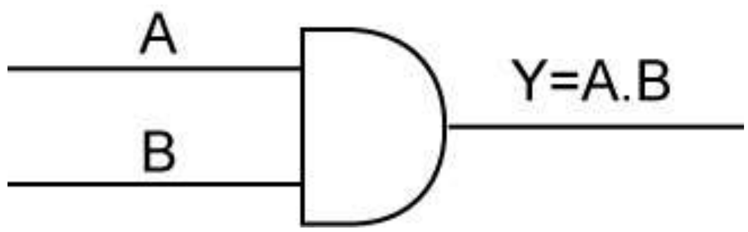
A	Y=NOT A
0	1
1	0

NOT Gate:



AND Gate

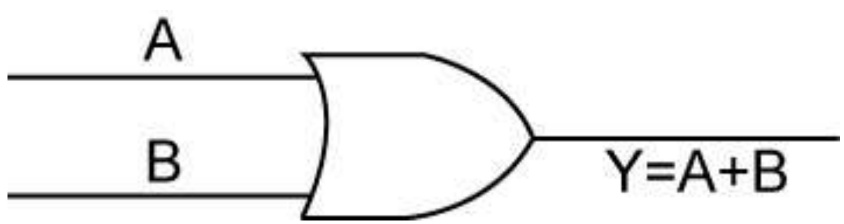
- The AND gate is a logic circuit that has two or more inputs and a single output
- The operation of the gate is such that the output of the gate is a binary 1 if and only if all inputs are binary 1
- Similarly, if any one or more inputs are binary 0, the output will be binary 0.



A	B	Y=A AND B
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

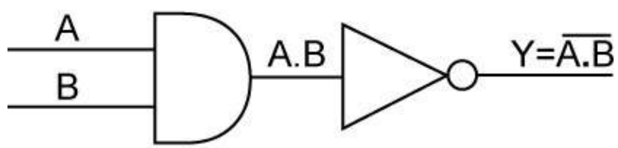
- The OR gate is another basic logic gate
- Like the AND gate, it can have two or more inputs and a single output
- The operation of OR gate is such that the output is a binary 1 if any one or all inputs are binary 1 and the output is binary 0 only when all the inputs are binary 0



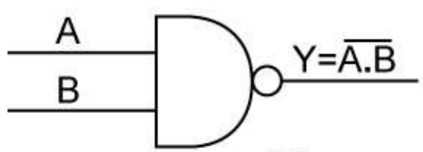
A	B	Y=A OR B
0	0	0
0	1	1
1	0	1
1	1	1

NAND Gate

- The term NAND is a contraction of the expression NOT-AND gate
- A NAND gate, is an AND gate followed by an inverter
- The algebraic output expression of the NAND gate is Y = A.B



(i)

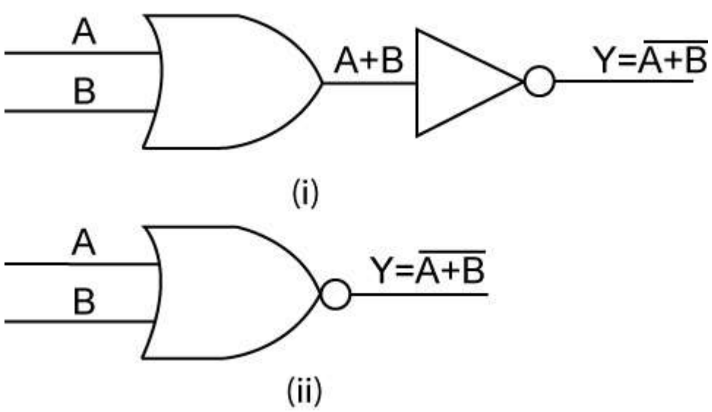


(ii)

A	B	Y= A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

- The term NOR is a contradiction of the expression NOT-OR
- A NOR gate, is an OR gate followed by an inverter
- The algebraic output expression of the NOR gate is Y = A + B



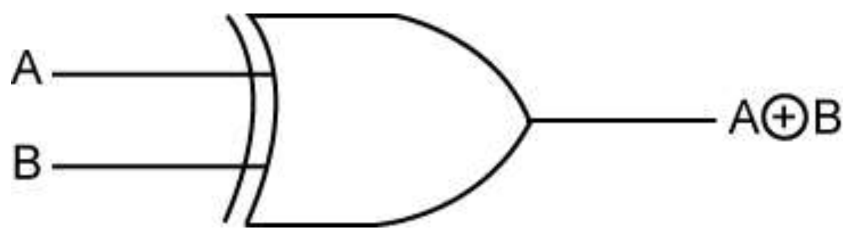
A	B	Y= A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

EX-OR and EX-NOR Gates

- EX-OR and EX-NOR are digital logic circuits that may use two or more inputs
- EX-NOR gate returns the output opposite to EX-OR gate
- EX-OR and EX-NOR gates are also denoted by XOR and XNOR respectively.

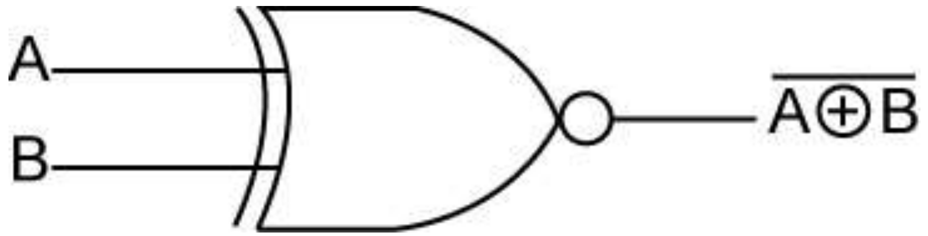
EXOR Gate

- The Ex-OR (Exclusive- OR) gate returns high output with one of two high inputs (but not with both high inputs or both low inputs)
- For example, if both the inputs are binary 0 or 1, it will return the output as 0. Similarly, if one input is binary 1 and another is binary 0, the output will be 1 (high)
- The operation for the Ex-OR gate is denoted by encircled plus symbol
- The Ex-OR operation is widely used in digital circuits.
- The algebraic output expression of the Ex-OR gate is $Y = A \oplus B = \bar{A}B + A\bar{B}$



EXNOR Gate

- The Ex-NOR (Exclusive- NOR) gate is a circuit that returns low output with one of two high inputs (but not with both high inputs)
- For example, if both the inputs are binary 0 or 1, it will return the output as 1. Similarly, if one input is binary 1 and another is binary 0, the output will be 0 (low)
- The symbol for the Ex-NOR gate is denoted by encircled plus symbol which inverts the binary values
- The algebraic output expression of the Ex-NOR gate is Y = $\overline{A \oplus B}$



Applications of Logic Gates

- The following are some of the applications of Logic gates:
  - Build complex systems that can be used to different fields such as
    - Genetic engineering,
    - Nanotechnology,
    - Industrial Fermentation,
    - Metabolic engineering and
    - Medicine
  - Construct multiplexers, adders and multipliers.
  - Perform several parallel logical operations
  - Used for a simple house alarm or fire alarm or in the circuit of automated machine manufacturing industry

1.2 Binary Numbers (1-6)

In general, a number expressed in a base-r system has coefficients multiplied by powers of r:

$$a_n.r^n + a_{n-1}.r^{n-1} + \ldots\ldots. + a_2.r^2 + a_1.r + a_0 + a_{-1}.r^{-1} + a_{-2}.r^{-2} + \ldots\ldots\ldots. + a_{-m}.r^{-m}$$

- r is called **base** or **radix**.
- a_j ranges in thr value from 0 to r-1

1.2 Binary Numbers (3-6)

Numbering System	Base	Symbols
Decimal	10	0, 1,2, 3, 4, 5, 6, 7, 8, 9
Binary	2	0, 1
Octal	8	0, 1,2, 3, 4, 5, 6, 7
Hexadecimal	16	0, 1,2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

1.2 Binary Numbers (2-6)

Example:

$$(127.4)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$$

$$(1101)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (13)_{10}$$

1.2 Binary Numbers (2-6)

Example:

Is:

$$(13)_{10} = (13)_8 \text{ ؟}$$

1.2 Binary Numbers (4-6)

Arithmetic Operations

Follow the same rules of as for decimal numbers

Addition

Carries	00000
Augend	01100
Addend	+ 10001
Sum	11101

Carries	101100
Augend	10110
Addend	+ 10111
Sum	101101

1.2 Binary Numbers (5-6)

Arithmetic Operations

Subtraction

Borrows	00000
Minuend	10110
Subtrahend	- 10010
Difference	00100

Borrows	00110
Minuend	10110
Subtrahend	- 10011
Difference	00011

Borrows	00110	00110
Minuend	10011	11110
Subtrahend	- 11110	- 10011
Difference		-01011

1.2 Binary Numbers (6-6)

Arithmetic Operations

Multiplication

Multiplicand	1011
Multiplier	X 101
	1011
	0000
	1011
Product	110111

1.3 Number-Base Conversions (1-6)

Base r – to – Decimal Conversion

Rule:

$$a_n.r^n + a_{n-1}.r^{n-1} + ... + a_2.r^2 + a_1.r + a_0 + a_{-1}.r^{-1} + a_{-2}.r^{-2} + ... + a_{-m}.r^{-m}$$

Integral part → $a_n.r^n + a_{n-1}.r^{n-1} + ... + a_2.r^2 + a_1.r + a_0$

Fractional part → $a_{-1}.r^{-1} + a_{-2}.r^{-2} + ... + a_{-m}.r^{-m}$

Result → **Integral part** . **Fractional part**

1.3 Number-Base Conversions (2-6)

Base r – to – Decimal Conversion

Example:

Find the decimal equivalent ofthe binary number $(1001.0101)_2$.

Integral part

$1001 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8+0+0+1=9$

Fractional part

$.0101 = 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 0+.25+0+0.0625=0.3125$

Result

$(9.3125)_{10}$

1.3 Number-Base Conversions (3-6)

Base r – to – Decimal Conversion

Example:

$(1010.011)_2 = 2^3 + 2^1 + 2^{-2} + 2^{-3} = (10.375)_{10}$

$(127.4)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$

$(1101)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (13)_{10}$

1.3 Number-Base Conversions (4-6)

Decimal – to – Base r Conversion

Rule:

Convert each part differently.

Integral part →
- **Divide** number & its quotients by r.
- Accumulate **reminders.**

Fractional part →
- **Multiply** number & its quotients by r.
- Accumulate **integers.**

Result → **Integral part** . **Fractional part**

1.3 Number-Base Conversions (5-6)

Decimal – to – Base r Conversion

Example:

Find the binary equivalent of the decimal number $(41)_{10}$.

Divide by 2

Integer	Reminder
41	
20	1
10	0
5	0
2	1
1	0
0	1

Result

$(101001)_2$

1.3 Number-Base Conversions (6-6)

Decimal – to – Base r Conversion

Example:

Find the binary equivalent of the decimal number $(0.6875)_{10}$.

Multiply by 2

	Integer	Fraction
0.6875	1	0.3750
0.3750	0	0.7500
0.7500	1	0.5000
0.5000	1	0.0000

Result

$(0.1011)_2$

1.4 Octal and Hexadecimal Numbers (1-17)

Table 1.2
Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1.4 Octal and Hexadecimal Numbers (2-17)

Decimal-to-Octal Conversion

Example:

Find the octal equivalent of the decimal number $(153.513)_{10}$.

Divide by 8

Integer	Reminder
153	
19	1
2	3
0	2

Multiply by 8

	Integer	Fraction
0.513	4	0.104
0.104	0	0.832
0.832	6	0.656
0.656	5	0.248
0.248	1	0.984
0.984	7	0.872

Result

$(231.406517)_8$

1.4 Octal and Hexadecimal Numbers (3-17)

Octal-to-Decimal Conversion

Example:

Find the decimal equivalent ofthe octal number $(137.21)_8$.

Integral part

$137 = 1 \times 8^2 + 3 \times 8^1 + 7 \times 8^0 = 64+24+7=95$

Fractional part

$.21 = 2 \times 8^{-1} + 1 \times 8^{-2} = 0.265$

Result

$(95.265)_{10}$

1.4 Octal and Hexadecimal Numbers (4-17)

Decimal-to-Hexadecimal Conversion

Example:

Find the hexadecimal equivalent of the decimal number $(82.25)_{10}$.

Divide by 16

Integer	Reminder
82	
5	2
0	5

Multiply by 16

	Integer	Fraction
0.25	4	.0000

Result

$(52.4)_{16}$

1.4 Octal and Hexadecimal Numbers (5-17)

Hexadecimal-to-Decimal Conversion

Example:

Find the decimal equivalent ofthe hexadecimal number $(1E0.2A)_{16}$.

Integral part → $1E0 = 1 \times 16^2 + 14 \times 16^1 + 0 \times 16^0 = 256+224+0=480$

Fractional part → $.2A = 2 \times 16^{-1} + 10 \times 16^{-2} = 0.164$

Result → $\mathbf{(480.164)_{10}}$

1.4 Octal and Hexadecimal Numbers (6-17)

Binary–Octal and Octal–Binary Conversions

Binary – Octal Rule:

- Divide **integral part** into ***three bits*** starting from ***right*** integral bit.
- Divide **fractional part** into ***three bits*** starting from ***left*** fractional bit.
- Adding zero's if necessary.

1.4 Octal and Hexadecimal Numbers (7-17)

Binary–Octal and Octal–Binary Conversions

Example:

Find the octal equivalent ofthe binary number $(1110100.0100111)_2$.

001 110 100 . 010 011 100

$(1\ 6\ 4\ .\ 2\ 3\ 4)_8$

1.4 Octal and Hexadecimal Numbers (8-17)

Binary–Octal and Octal–Binary Conversions

Octal – Binary Rule:

BOTH integral and fractional parts:

- Convert each **digit** seperately into binary of ***three bits***

1.4 Octal and Hexadecimal Numbers (9-17)

Binary–Octal and Octal–Binary Conversions

Example:

Find the binary equivalent ofthe octal number $(374.26)_8$.

3	7	4	.	2	6
011	111	100	.	010	110

$(11\ 111\ 100\ .\ 010\ \ 11)_2$

1.4 Octal and Hexadecimal Numbers (10-17)

Binary–Hex and Hex–Binary Conversions

Binary – Hex Rule:

- Divide ***integral part*** into ***four bits*** starting from ***right*** integral bit.
- Divide ***fractional part*** into ***four bits*** starting from ***left*** fractional bit.
- Adding zero's if necessary.

1.4 Octal and Hexadecimal Numbers (11-17)

Binary–Hex and Hex–Binary Conversions

Example:

Find the hexdecimal equivalent ofthe binary number $(1011001110.011011101)_2$.

0010 1100 1110 . 0110 1110 1000

$(2\ C\ E\ .\ 6\ E\ 8)_{16}$

1.4 Octal and Hexadecimal Numbers (12-17)

Binary–Hex and Hex–Binary Conversions

Hex – Binary Rule:

BOTH integral and fractional parts:

- Convert each **digit** seperately into binary of ***four bits***

1.4 Octal and Hexadecimal Numbers (13-17)

Binary–Hex and Hex–Binary Conversions

Example:

Find the binary equivalent ofthe hexadecimal number $(17E.F6)_{16}$.

1 7 E . F 6

0001 0111 1110 . 1111 0110

$(1\ 0111\ 1110\ .\ 1111\ 011)_2$

1.4 Octal and Hexadecimal Numbers (14-17)

Hex–Octal and Octal–Hex Conversions

Hex – Octal Rule:

- Convert Hex number into binary.
- Convert the result binary number into octal.

OR

- Convert Hex number into decimal.
- Convert the result decimal number into octal.

1.4 Octal and Hexadecimal Numbers (15-17)

Hex–Octal and Octal–Hex Conversions

Example:

Find the Hex equivalent ofthe octal number $(762.013)_8$.

$$\begin{array}{cccccccc} 7 & 6 & 2 & . & 0 & 1 & 3 \end{array}$$

Binary

$$(0001\ 1111\ 0010\ .\ 0000\ 0101\ 1000)_2$$

Hex

$$(1\ F\ 2\ .\ 0\ 5\ 8)_{16}$$

1.4 Octal and Hexadecimal Numbers (16-17)

Hex–Octal and Octal–Hex Conversions

Octal – Hex Rule:

- Convert Octal number into binary.
- Convert the result binary number into Hex.

OR

- Convert Octal number into decimal.
- Convert the result decimal number into Hex.

1.4 Octal and Hexadecimal Numbers (17-17)

Hex–Octal and Octal–Hex Conversions

Example:

Find the octal equivalent ofthe Hex number $(2F.C4)_{16}$.

2 F . C 4

Binary

$(000\ 101\ 111\ .\ 110\ 001\ 000)_2$

Octal

$(0\ 5\ 7\ .\ 6\ 1\ 0)_8$ → $(5\ 7\ .\ 6\ 1)_8$

1.5 Complements (1-9)

Complement's Types

Radix Complement
(r's complement)

Diminished radix Complement
((r-1)'s complement)

1.5 Complements (1-9)

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

Rule

- Given a number N in base r having n digits, the (r- 1)'s complement of N is defined as $(r^n - 1) - N$.

Examples

- 9's complement of 546700 is 999999 – 546700= 453299
- 1's complement of 1011000 is 0100111.

Note

- The (r-1)'s complement of ***octal*** or ***hexadecimal*** numbers is obtained by subtracting each digit from **7** or **F** (decimal 15), respectively.

1.5 Complements (2-9)

Radix Complement

Rule

- Given a number N in base r having n digit, the r's complement of N is defined as (r^n -N) for N ≠0 and as 0 for N =0 .

OR

- Adding 1 to (*r-1*)'s complement.

Examples

- The **10's complement** of 012398 is 987602
- The **10's complement** of 246700 is 753300
- The **2's complement** of 1011000 is 0101000

1.5 Complements (2-9)

Note:

- The complement of the complement restores the number to its original value.

1.5 Complements (3-9)

Subtraction with Complement

The subtraction of two n-digit <u>unsigned</u> numbers $M - N$ in base r can be done as follows:

- $M + (r^n - N)$, note that $(r^n - N)$ is r's complement of N.
  - If (**$M \geq N$**), the sum will produce an end carry x, which can be discarded; what is left is the result M- N.
  - If (**$M < N$**), the sum does not produce an end carry and is (N - M). Take the r'x complement of the sum and place a negative sign in front.

1.5 Complements (4-9)

Example:

Using 10's complement subtract 72532 – 3250.

M = 72532
10's complement of N = 96750
sum = 169282
Discard end carry.
answer: 69282

1.5 Complements (5-9)

Example:

Using 10's complement subtract 3250 - 72532 .

M = 03250

10's complement of N = 27468

sum = 30718

The answer is –(10's complement of 30718) = -69282

1.5 Complements (6-9)

Example:

Using 2's complement subtract 1010100 – 1000011 .

M = 1010100

N = 1000011, 2's complement of N = 0111101

1010100 + 0111101 = 10010001

Discard end carry.

answer: 0010001

1.5 Complements (7-9)

Example:

Using 2's complement subtract 1000011 – 1010100.

M = 1000011

N = 1010100, 2's complement of N = 0101100

1000011 + 0101100 = 1101111

The answer is –(2's complement of 1101111) = -0010001

1.5 Complements (8-9)

Example:

Using 1's complement subtract 1010100 - 1000011.

M = 1010100
N = 1000011, 1's complement of N = 0111100

1010100 + 0111100 = 10010000
end-around carry = + 1

The answer is: 0010001

1.5 Complements (9-9)

Example:

Using 1's complement subtract 1000011 - 1010100.

M = 1000011

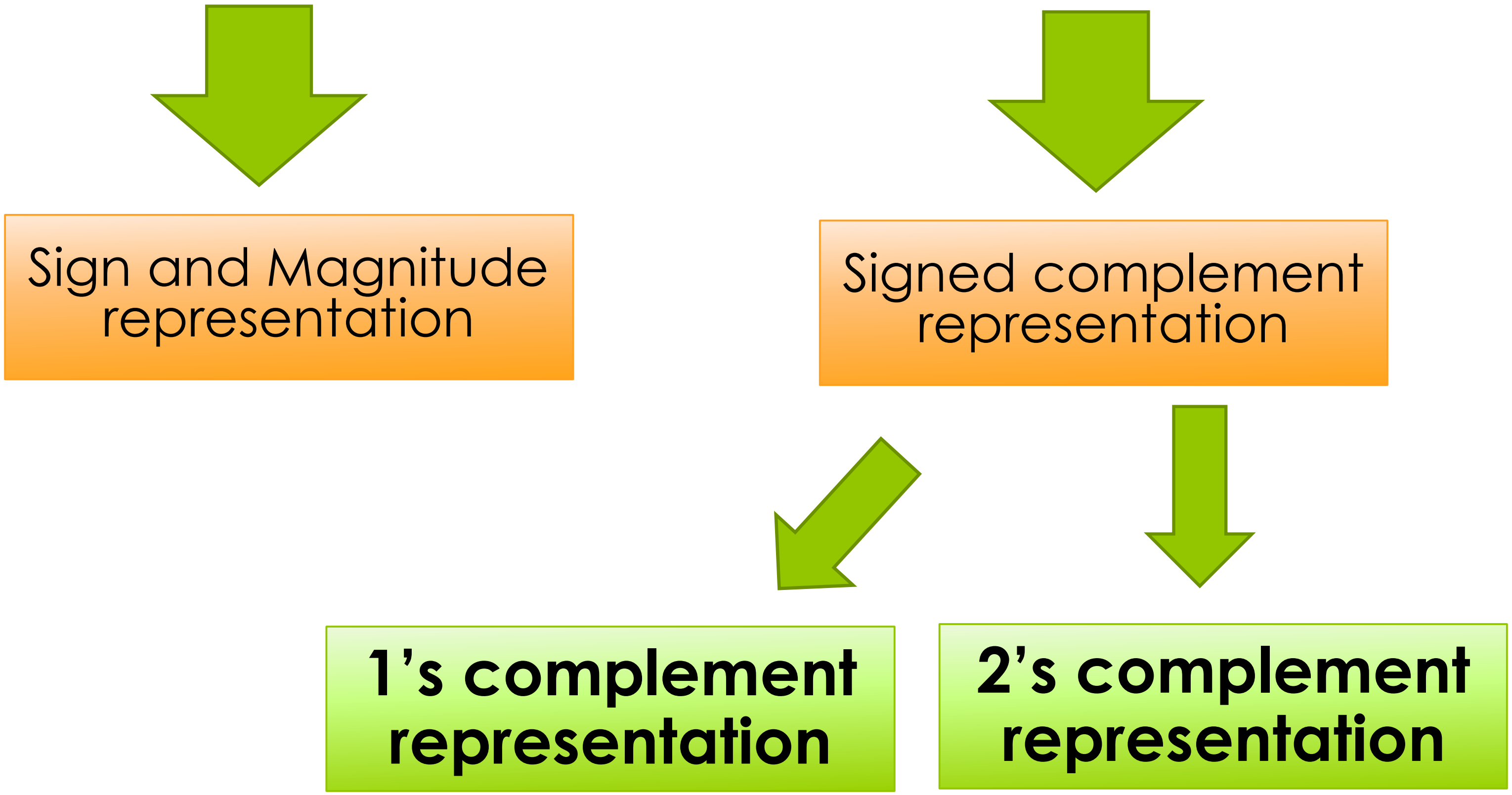
N = 1010100, 1's complement of N = 0101011

1000011 + 1010100 = 1101110

No end-around carry → Take the 1's complement of the result with negetive sign.

The answer is: -0010001

1.6 Signed Binary Numbers (1-10)



1.6 Signed Binary Numbers (2-10)

Notes:

- The previous representation (***unsigned numbers***) are the ***same*** for ***positive*** numbers and ***different*** for ***negative*** numbers.
- For a ***unsigned binary number*** the most significant bit is part ***of the number.***
- For a ***signed binary number*** the most significant bit is used for representing the ***sign of the number*** and the rest of the bits represent the ***number***.

We use 0 for positive numbers and 1 for negative numbers.

1.6 Signed Binary Numbers (3-10)

Example:

Represent +76

$(76)_{10} = (1001100)_2$	→	Unsigned number
$(+76)_{10} = (\mathbf{0}1001100)_2$	→	Sign & Magnitude
$(+76)_{10} = (\mathbf{0}0110011)_2$	→	1's Complement
$(+76)_{10} = (\mathbf{0}1001101)_2$	→	2's Complement

1.6 Signed Binary Numbers (4-10)

Negative numbers:

1‘Complement

obtained by flipping all bits of the positive binary number

2‘Complement

obtained by adding 1 to the 1’s Complement.

OR:

by flipping bits of the positive binary number after the first one from the right

1.6 Signed Binary Numbers (5-10)

Negative numbers:

Sign & Magnitude

obtained by changing the **left most bit** of the positive binary number to 1.

Example:

Represent -76

$(-76)_{10} = (\mathbf{1}1001100)_2$ → Sign & Magnitude

$(-76)_{10} = (\mathbf{1}0110011)_2$ → 1's Complement

$(-76)_{10} = (\mathbf{1}1001101)_2$ → 2's Complement

1.6 Signed Binary Numbers (8-10)

Arithmetic Addition:

Without Comprison

$$\begin{array}{rr r}
+ & 06 & +00000110 \\
+ & 13 & +00001101 \\
\hline
+ & 19 & 00010011
\end{array}
\qquad
\begin{array}{rr r}
- & 06 & +11111010 \\
+ & 13 & +00001101 \\
\hline
+ & 07 & 00000111
\end{array}$$

$$\begin{array}{rr r}
+ & 06 & +00000110 \\
- & 13 & +11110011 \\
\hline
- & 07 & 11111001
\end{array}
\qquad
\begin{array}{rr r}
- & 06 & +11111010 \\
- & 13 & +11110011 \\
\hline
- & 19 & 11101101
\end{array}$$

1.6 Signed Binary Numbers (9-10)

Arithmetic Subtraction:

Rule

- Change the artithmetic operation into addition.
- Change the sign of the subtrahend (***positive to negative or negative to positive***).

(+/-) A – (+B)= (+/-) A + (-B)
(+/-) A – (-B)= (+/-) A + (+B)

1.6 Signed Binary Numbers (10-10)

Arithmetic Subtraction:

Example

- (-6) – (-13)= +7
- **<u>In binary</u>**: (1111010 – 11110011)= (1111010 +00001101) =100000111
- After removing the carry out the result will be : 00000111

1.7 Binary Codes (3-24)

BCD

- Stands for **binary-coded decimal**.
- Represents the **10 decimal** digits.
- Each **digit** is represented in **4 bits.**
- Makes **16** possible combinations.
- **6** are unassigned.
- A number with ***k decimal digits*** will require ***4k bits*** in BCD.
- **Weight** (8,4,2,1.)

Table 1.4
Binary-Coded Decimal (BCD)

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

1.7 Binary Codes (4-24)

BCD

Example

Represent $(945)_{10}$ in BCD

3 decimal digits will require (3*4) = 12 bits.
Each decimal digit is represented in 4 bits.

9	1001
4	0100
5	0101

Result $(1001\ 0100\ 0101)_{BCD}$

1.7 Binary Codes (5-24)

BCD Addition

Notes

- Same as adding two decimal numbers.
- Sum cannot be ***greater*** than 9, such as: 9+9+1=19 with 1 is being a previous carry.
- If the result value was ***greater*** than $(1001)_2$=(9) or if the result was more than ***4*** digits.
  - Then it is an **invalid** BCD digit.
  - Add $(0110)_2$=(6) to convert it to correct digit.
  - A carry will be produced

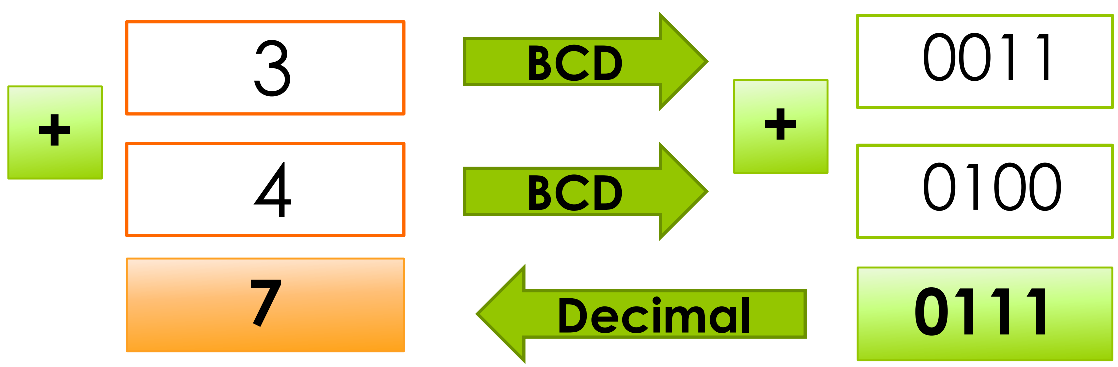
Result more than 4 digit is greater than 9(1001)

1.7 Binary Codes (6-24)

BCD Addition

Example

Evaluate <u>(3+4)</u> in BCD System



1.7 Binary Codes (7-24)

BCD Addition

Example

Evaluate <u>(4+8)</u> in BCD System

	Decimal		BCD
+	4	BCD →	0100
	8	BCD →	1000
	12		**1100**

ERROR

+ 0110

We must add 6 (0110) to the result

0001 0010

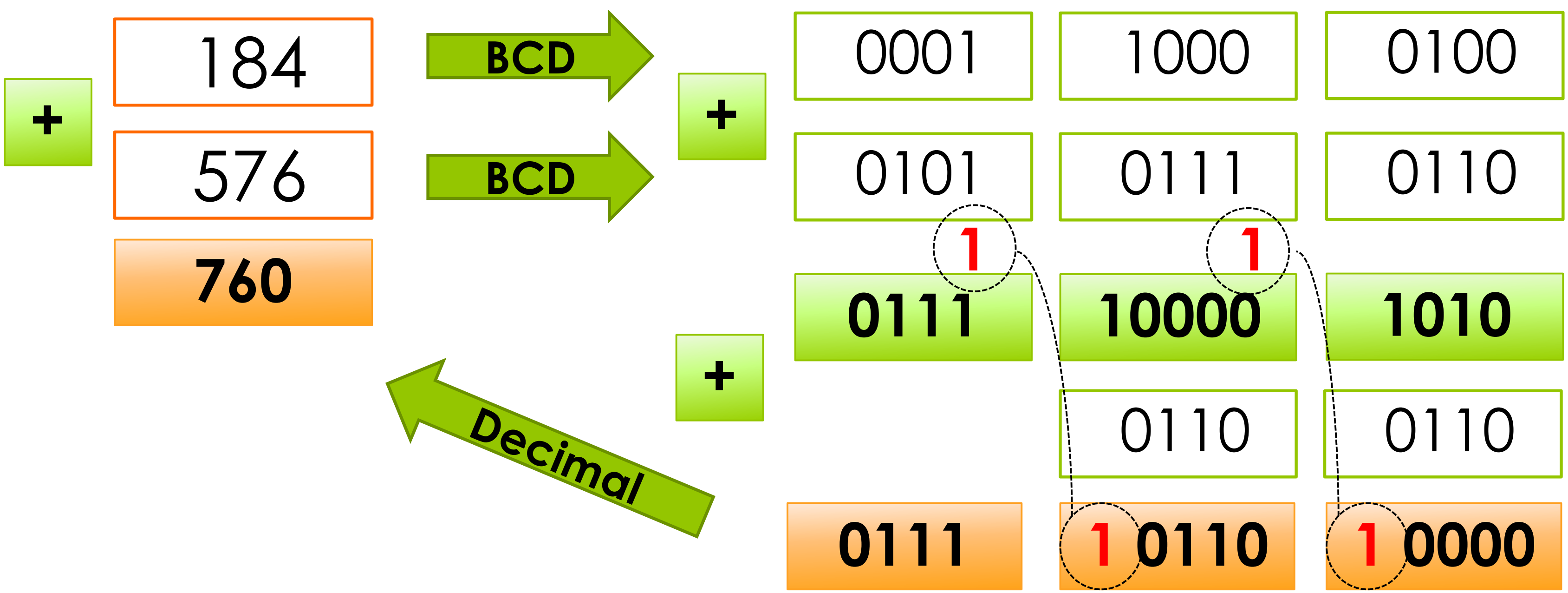
Decimal → **12**

1.7 Binary Codes (8-24)

BCD Addition

Example

Evaluate (184+576) in BCD System



1.7 Binary Codes (12-24)

Other Decimal codes

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1[illegible]	1010	1001
8	1000	1[illegible]	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
Unused	1011	0110	0001	0010
bit	1100	0111	0010	0011
combi-	1101	1000	1101	1100
nations	1110	1001	1110	1101
	1111	1010	1111	1110

1.7 Binary Codes (13-24)

Excess -3 (ex-3)

- Another system to represent a ***decimal number***.
- Like (**BCD**) in the way of representing number (***each digit is represented in 4 bits***)

Except that **: each digit is firstly incremented by three**

1.7 Binary Codes (14-24)

Excess -3 (ex-3)

Example

Represent $(945)_{10}$ in ex-3

9	**12**	1100
4	**7**	0111
5	**8**	1000

Result $(1100\ 0111\ 1000)_{ex\text{-}3}$

1.7 Binary Codes (15-24)

Gray Code

- Output data of many physical systems are quantities that are continuous.
- Such data must be converted into digital form before they are applied to a digital *system using* ***analog-to-digital converter***.
- Such converted data represented using **Gray code**.
  - Used in applications in which the normal sequence of binary numbers may produce an error or ambiguity during the transition from one number to the next.
  - Because only 1bit in the code group changes in going from one number to the next.

1.7 Binary Codes (16-24)

Gray Code

One-bit Gray code	Two-bit Gray code		Three-bit Gray code		Four-bit Gray code	
0	0	00	00	000	000	0000
1	1	01	01	001	001	0001
	1	11	11	011	011	0011
	0	10	10	010	010	0010
			10	110	110	0110
			11	111	111	0111
			01	101	101	0101
			00	100	100	0100
					100	1100
					101	1101
					111	1111
					110	1110
					010	1010
					011	1011
					001	1001
					000	1000

1.7 Binary Codes (17-24)

Gray Code

Gray Code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

1.7 Binary Codes (18-24)

ASCII Character Code

- ASCII : **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- Used to represent characters , Symbols , …
- Consists of 7-bits (***to represent 128 character***).
- A bit can be added to produce a total of eight bits (byte).
  - Used for other purposes.

1.7 Binary Codes (22-24)

ASCII Character Code

Error Detecting Code

- Detecting errors in data communication and processing.
- Eight bit added to ASCII code.
- A ***parity bit*** is an extra- bit included with a message lo make the total number of 1's either ***even*** or ***odd***.

Example:

	even parity	odd parity
ASCII (A) 1000001	01000001	11000001
ASCII (T) 1010100	11010100	01010100

1.7 Binary Codes (23-24)

ASCII Character Code

Error Detecting Code

Parity bit

Odd parity		Even parity	
Message	P	Message	P
0000	1	0000	0
0001	0	0001	1
0010	0	0010	1
0011	1	0011	0
0100	0	0100	1
0101	1	0101	0
0110	1	0110	0
0111	0	0111	1
1000	0	1000	1
1001	1	1001	0
1010	1	1010	0
1011	0	1011	1
1100	1	1100	0
1101	0	1101	1
1110	0	1110	1
1111	1	1111	0

1.7 Binary Codes (24-24)

ASCII Character Code

Error Detecting Code

- The eight-bit character that include parity bits are transmitted to their destination.
- The parity of each character is then checked at the receiving end.
- If the parity of the received character is not even. then at least one bit has changed value during transmission.
- Detects one, three or any odd combination of errors in each character that is transmitted.
- Other detection codes may be needed to take care of that possibility