# Digital Systems

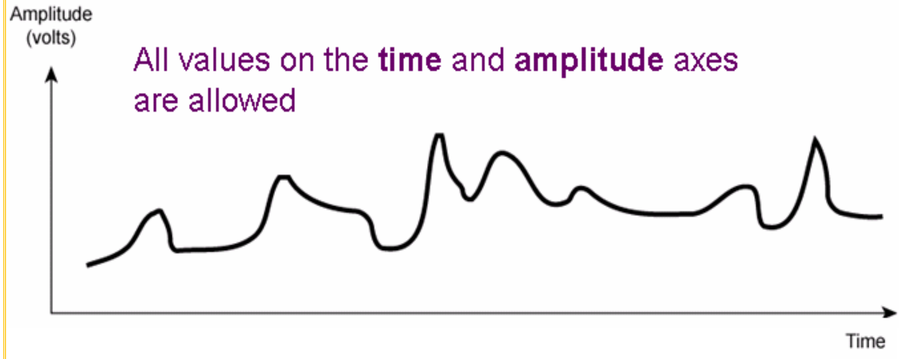
## Section 2

## Chapter (1)

# Analog vs. Digital

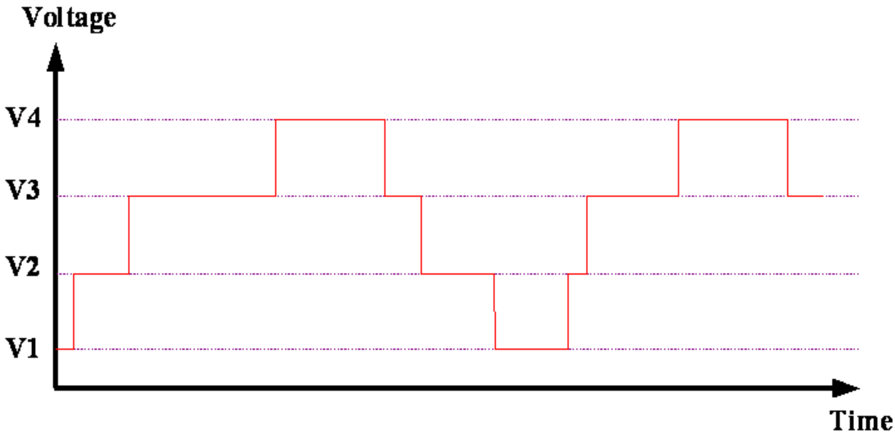
- World around us is predominantly **Analog**
  - **Analog** = <u>Continuous</u>
    - Change smoothly and gradually over **time.**
    - Assume a <u>continuous</u> (infinite) range of **amplitudes.**

*- Earth's movement - Speech signal - Body temperature*




- Binary is a special case of Digital
  - **Digital** = <u>Discrete</u>
    - Takes only a limited (finite) set of "**Discrete**" values.
    - Changes abruptly in **time** by "Jumping" between levels.

*- Position of a switch - The Alphabet - DNA sequence*



# Digital over Analog

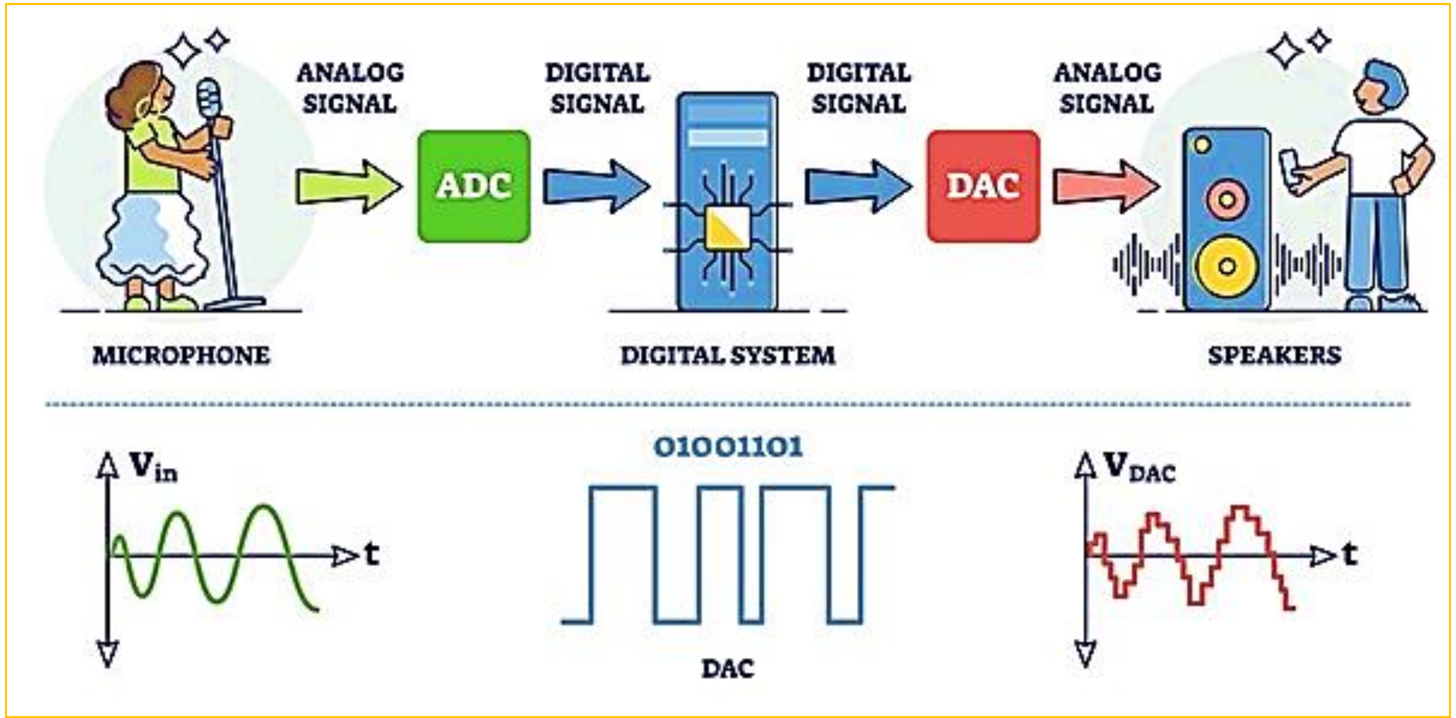
- ✓ **Easier to Design:** Digital circuits are simpler because they handle a **limited set** of values (e.g. binary), making design more straightforward compared to analog systems.
- ✓ **Error Tolerance:** Digital circuits are more resilient to noise and drift, leading to **lower** error rates and higher reliability.
- ✓ **Commercial Advantage:** In VLSI (Very Large Scale Integration), digital circuits are more **cost-effective** and widely available.
- ✓ **Digital Superiority:** Storing, encrypting, compressing, and communicating data is far **more efficient** with digital systems.

**"But the natural world is analog... So, we need to convert!"**

**"ADC and DAC: Bridging the gap between continuous analog signals and discrete digital data."**

- Analog-to-Digital Converters (ADC):
  Used to transform raw analog signals into digital form for processing.

- Digital-to-Analog Converters (DAC):
  Used to regenerate analog signals from digital data.

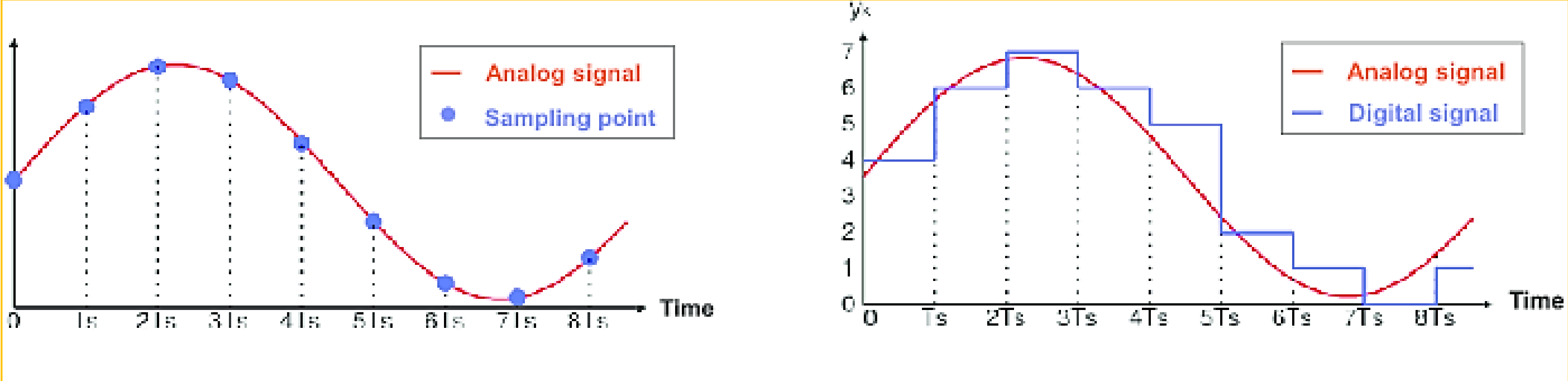


ENCS
2340

# Convert Analog Signals to Digital Samples

1. **Sampling in Time:**
   Impossible to manage infinite signal values on the **time** axis, so we ignore the signal between samples.

2. **Quantization in Amplitude:**
   Impossible to handle infinite **amplitude** values, so we approximate the sample to the nearest value from a finite set of levels.




(a) (b)

ENCS 2340

## Abstract Representations

- Arithmetic Values: **0, 1**
- Logic Levels: **True, False**
- States: **ON, OFF**

## Physical Representations

- In an IC (e.g. in a microprocessor): **Voltage**
- In a Dynamic memory (DRAM): **Electric Charge**
- On a Hard Disk: **Magnetization Direction**
- On a CD: **Surface pits for laser interference**

# Number Systems

4692

ENCS 2340

4692.89

# Decimal Number System



$$
\begin{aligned}
4692.89 &= 4000 + 600 + 90 + 2 + 0.8 + 0.09 \\
&= 4 \times 1000 + 6 \times 100 + 9 \times 10 + 2 \times 1 + 8 \times 0.1 + 9 \times 0.01 \\
&= \underbrace{4}_{a_3} \times 10^3 + \underbrace{6}_{a_2} \times 10^2 + \underbrace{9}_{a_1} \times 10^1 + \underbrace{2}_{a_0} \times 10^0 + \underbrace{8}_{a_{-1}} \times 10^{-1} + \underbrace{9}_{a_{-2}} \times 10^{-2} \\
&= a_3 \times r^3 + a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0 + a_{-1} \times r^{-1} + a_{-2} \times r^{-2}
\end{aligned}
$$

where $a_n$'s are the coefficients of units, tens, hundreds, thousands, etc.

$$a_3 = 4, a_2 = 6, a_1 = 9, a_0 = 2, a_{-1} = 8, a_{-2} = 9, \text{and}$$
$$r = 10$$

---

In general, a number can be written as:

$$\cdots + a_3 \times r^3 + a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0 + a_{-1} \times r^{-1} + a_{-2} \times r^{-2} + \cdots$$

- $r$ is called the *radix* or *base*
- $a_n$'s are called the coefficients
- $a_n$'s range from $0$ to $r - 1$
- In decimal number system, $r = 10$ and $a_n$'s range from $0$ to $9$

- How do we count in a decimal number system?
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- What comes after 9?
  We have run out of numbers...

- Remember units, tens, hundreds, thousands, etc.?
  i. We write **1** at the position of **tens** and a **0** at the position of **units** i.e., **10** and then we continue by increasing the **units** position one-by-one until we reach **19**

  ii. Once at **19** we change the number at **tens** position and then start changing the number at **units** position

  iii. When we reach at **99**, we have to put a **1** at **hundreds** position and then continue in the manner described above and **the counting goes on**...

# Octal Number System



- If the radix (r = **8**) → we get the **Octal number system**
- Since r = 8, the coefficients $a_n$'s will range from 0 to (8-1) → **0 to 7**
  - 470, 7501, 2636, 777 (**Valid**)
  - 870, 7901, 2838, 779 (**Invalid**)
- We can count in Octal number system just like we do in decimal number system:
  0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22, 23, · · · , 65, 66, 67, 70, 71, 72, 73, 74, 75, 76, 77, · · ·, 100, 101, 102, 103, 104,105, 106, 107, 110, · · ·

- If we have an octal number, is it possible to find its equivalent decimal number?
  - **Example**: Let's say we have an octal number **107**, we may represent it as:

$$a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0$$

$$1 \times 8^2 + 0 \times 8^1 + 7 \times 8^0 = 71$$

  - 107 (octal) = 71 (decimal)

- To avoid confusion, we use the radix/base along with the numbers → $\mathbf{(107)_8 = (71)_{10}}$
- What about $\mathbf{(1000)_8}$ and $\mathbf{(999)_8}$ ? $(1000)_8 = \mathbf{(512)_8}$ , (999) is an **Invalid** Octal Number.

- Decimal equivalent of Octal numbers with radix points will have a general form:

$$\cdots + a_3 \times r^3 + a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0 + a_{-1} \times r^{-1} + a_{-2} \times r^{-2} + \cdots$$

$$\cdots + a_3 \times 8^3 + a_2 \times 8^2 + a_1 \times 8^1 + a_0 \times 8^0 + a_{-1} \times 8^{-1} + a_{-2} \times 8^{-2} + \cdots$$

$$\cdots + a_2 \times 512 + a_2 \times 64 + a_1 \times 8 + a_0 \times 1 + a_{-1} \times 0.125 + a_{-2} \times 0.015625 + \cdots$$

$(107.73)_8 = (????)_{10}$

## Octal Number System



$(107.73)_8 = (????)_{10}$

$$\Rightarrow 1 \times 8^2 + 0 \times 8^1 + 7 \times 8^0 + 7 \times 8^{-1} + 3 \times 8^{-2}$$

$$\Rightarrow 1 \times 64 + 0 \times 8 + 7 \times 1 + 7 \times 0.125 + 3 \times 0.015625$$

$$\Rightarrow 64 + 0 + 7 + 0.875 + 0.046875$$

$= (71.921875)_{10}$

- If the radix (r = **2**) → we have the **binary number system**

- Since r = 2, the coefficients $a_n$'s will be either 0 or (2-1) → **0 or 1**.
  - **0 , 1 :** called **bi**nary digi**ts** or **bits**
  - 101, 1111, 1010, 110 (**Valid**)
  - 201, 1311, 1212, 115 (**Invalid**)

- Decimal equivalent of a binary number will have a general form →

$$\cdots + a_3 \times r^3 + a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0$$

$$\cdots + a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$$

$$\cdots + a_3 \times 8 + a_2 \times 4 + a_1 \times 2 + a_0 \times 1$$

- Since $a_n$'s could be either 0 or 1, the decimal equivalent of a binary number is simply a **sum** of **powers of 2.**

*Powers of Two*

| *n* | $2^n$ | *n* | $2^n$ | *n* | $2^n$ |
|---|---|---|---|---|---|
| 0 | 1 | 8 | 256 | 16 | 65,536 |
| 1 | 2 | 9 | 512 | 17 | 131,072 |
| 2 | 4 | 10 | 1,024 (1K) | 18 | 262,144 |
| 3 | 8 | 11 | 2,048 | 19 | 524,288 |
| 4 | 16 | 12 | 4,096 (4K) | 20 | 1,048,576 (1M) |
| 5 | 32 | 13 | 8,192 | 21 | 2,097,152 |
| 6 | 64 | 14 | 16,384 | 22 | 4,194,304 |
| 7 | 128 | 15 | 32,768 | 23 | 8,388,608 |

- Let's see some **Examples:**

➡ What is the decimal equivalent of binary number 11?

$$
\begin{aligned}
(11)_2 &= 1 \times 2^1 + 1 \times 2^0 \\
&= 1 \times 2 + 1 \times 1 \\
&= 2 + 1 \\
(11)_2 &= (3)_{10}
\end{aligned}
$$

➡ What is the decimal equivalent of $(101)_2$?

$$
\begin{aligned}
(101)_2 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
&= 1 \times 4 + 0 \times 1 + 1 \times 1 \\
&= 4 + 1 \\
(101)_2 &= (5)_{10}
\end{aligned}
$$

## More Examples:

➡ What is the decimal equivalent of binary number 1011?

$$
\begin{aligned}
(1011)_2 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
&= 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 \\
&= 8 + 2 + 1 \\
(1011)_2 &= (11)_{10}
\end{aligned}
$$

543210

➡ What is the decimal equivalent of $(101101)_2$?

32 8 4 1

= 32+8+4+1
= 45

**Always use the simpler method**

- What if we have a binary number with radix point such as: $(\mathbf{1011.11})_2$
- Remember, the general form of decimal equivalent of a binary number can be written as:

$$\Rightarrow \cdots + a_3 \times r^3 + a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0 + a_{-1} \times r^{-1} + a_{-2} \times r^{-2} + \cdots$$

$$\Rightarrow \cdots + a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0 + a_{-1} \times 2^{-1} + a_{-2} \times 2^{-2} + \cdots$$

- Therefore,

$$\begin{aligned}(1011.11)_2 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} \\ &= 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 + 1 \times 0.5 + 1 \times 0.25 \\ &= 8 + 2 + 1 + 0.5 + 0.25 \\ &= (11.75)_{10}\end{aligned}$$

## Before we move on, let's summarize



- So far, we have seen three different number systems:
  - Binary: r = 2 with binary digits (bits) 0, 1
  - Octal: r = 8 with octal digits 0, 1, 2, 3, 4, 5, 6, 7
  - Decimal: r = 10 with digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- We can have more number systems. In fact, we can have **infinite** number of number systems as there are infinite possibilities for **r**.
- There is another system that is commonly used: **r = 16**

- If the radix (r = **16**) → we have the **Hexadecimal number system**
- Since r = 16, it requires **16** coefficients (digits):
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9... now what?
  - It seems we are short of numbers!!! what should we do now?
  - We will use **A, B, C, D, E, F** as the remaining **6** digits
- Counting in hexadecimal number system:
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Some Examples:
  1E43, 1018, 2AB5D, FCF, 0044, 5A4B3CDEF

What is the decimal equivalent of $(A30C)_{16}$ ?

$(A30C)_{16} = (????)_{10}$

$$
\begin{aligned}
(A30C)_{16} &= A \times 16^3 + 3 \times 16^2 + 0 \times 16^1 + C \times 16^0 \\
&= A \times 4096 + 3 \times 256 + 0 \times 16 + C \times 1 \\
&= 10 \times 4096 + 3 \times 256 + 0 \times 16 + 12 \times 1 \\
&= 41740 \\
(A30C)_{16} &= (41740)_{10}
\end{aligned}
$$

# Summary of Number Systems



| 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 |

# Summary of Number Systems



| System | General | Decimal | Binary | Octal | Hexadecimal |
|---|---|---|---|---|---|
| Radix (Base) | **r** | **10** | **2** | **8** | **16** |
| Digit Values | 0, 1, …, (r – 1) | 0, 1, …, 9 | 0, 1 | 0,…, 7 | 0, …, 15 |
| 5 | $r^5$ | 100,000 | 32 | 32,768 | 1,048,576 |
| 4 | $r^4$ | 10,000 | 16 | 4,096 | 65,536 |
| 3 | $r^3$ | 1,000 | 8 | 512 | 4,096 |
| 2 | $r^2$ | 100 | 4 | 64 | 256 |
| 1 | $r^1$ | 10 | 2 | 8 | 16 |
| 0 | $r^0$ | 1 | 1 | 1 | 1 |
| -1 | $r^{-1}$ | 0.1 | 0.5 | 0.125 | 0.0625 |
| -2 | $r^{-2}$ | 0.01 | 0.25 | | |
| -3 | $r^{-3}$ | 0.001 | 0.125 | | |
| -4 | $r^{-4}$ | 0.0001 | 0.0625 | | |
| -5 | $r^{-5}$ | 0.00001 | 0.03125 | $8^{-5}$ | $16^{-5}$ |

The three bases are Power of 2

## For any number system with radix **r**

e.g. r=7

- The number of possible digits equals **r**. — 7
- The general form is:

Number $A$ of $n$ integral digits and $m$ fractional

$$MSD \rightarrow \underbrace{a_{n-1}a_{n-2}\ldots a_2a_1a_0}_{\text{integral}} \bullet \underbrace{a_{-1}a_{-2}\ldots a_{-m}}_{\text{fractional}} \leftarrow LSD$$

MSD → Most Significant Digit
LSD → Least Significant Digit

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

- The smallest digit is **0** and the largest possible digit has a value of **(r-1)** — 0,1,.,.,.,5,6
- The **Largest** value that can be expressed in **n** integral digits is $(r^n - 1)$ — e.g. n=3, $7^3$-1 = $(666)_7$
- The **Largest** value that can be expressed in **m** fractional digits is $(1 - r^{-m})$ — e.g. m=3, $(1-7^{-3}) = (0.666)_7$
- The **Largest** value that can be expressed in **n** integral digits and **m** fractional digits is $(r^n - r^{-m})$ — $(666.666)_7$
- **Total** number of values representable in **n** digits is $r^n$ — $7^3$ : 000 → 666

*Act fast and snag a quick mark!*



$$(12x4)_r = (52)_r$$

*r = ⁺1 Mark*

# Number-Base Conversion

**General Case:**

Bases are not powers of a common number.

Go through **decimal** as an intermediate step

e.g. $(231)_5 \rightarrow (??)_8$ : $(231)_5 \rightarrow (??)_{10} \rightarrow (??)_8$

**Special Case:**

Bases are powers of a common number (e.g. **2**)

Use the **common number** (here is the **Binary**) as an intermediate step

e.g. $(635)_8 \rightarrow (??)_{16}$ : $(635)_8 \rightarrow (??)_2 \rightarrow (??)_{16}$

## Octal, Hexadecimal, & Binary

**Remember:**

$8 = 2^3 \rightarrow$ Base 8 (OCTAL System)

$16 = 2^4 \rightarrow$ Base 16 (HEXADECIMAL System)

**This means:**

- In OCT: each **3 binary** bits can be represented with a **single octal** digit
- In HEX: each **4 binary** bits can be represented with a **single hexadecimal** digit

| Octal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Binary | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

$(b_n \ldots b_5 b_4 b_3 b_2 b_1 b_0 \bullet b_{-1} b_{-2} b_{-3} b_{-4} \ldots)_2 \Rightarrow (?)_8$

Starting from the radix point and in both directions, group every 3 Binary Bits.

Replace the 3-Bit groups with equivalent Octal digits.

$(b_n \ldots \underbrace{\ldots b_4 b_3}_{\text{3-Bit}} \underbrace{b_2 b_1 b_0}_{\text{3-Bit}} \bullet \underbrace{b_{-1} b_{-2} b_{-3}}_{\text{3-Bit}} \underbrace{b_{-4} \ldots}_{\text{3-Bit}})_2$

| Hexadecimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Binary | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 |
| Hexadecimal | 8 | 9 | A | B | C | D | E | F |
| Binary | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |

$(b_n \ldots b_5 b_4 b_3 b_2 b_1 b_0 \bullet b_{-1} b_{-2} b_{-3} b_{-4} \ldots)_2 \Rightarrow (?)_{16}$

Starting from the radix point and in both directions, group every 4 Binary Bits.

Replace the 4-Bit groups with equivalent Hexadecimal digits.

$(b_n \ldots \underbrace{\ldots b_5 b_4}_{\text{4-Bit}} \underbrace{B_3 b_2 b_1 b_0}_{\text{4-Bit}} \bullet \underbrace{b_{-1} b_{-2} b_{-3} B_{-4}}_{\text{4-Bit}} \underbrace{b_{-5} b_{-6} \ldots}_{\text{4-Bit}})_2$

# Direct/Special Conversion



## Octal ⟷ Binary

$(1110010101.1011011)_2 \Rightarrow (?)_8$

$(\underbrace{001}_{1}\ \underbrace{110}_{6}\ \underbrace{010}_{2}\ \underbrace{101}_{5} \bullet \underbrace{101}_{5}\ \underbrace{101}_{5}\ \underbrace{100}_{4})_2 = (1625.554)_8$

$(37.2)_8 \Rightarrow (?)_2$

$(\underbrace{3}_{011}\ \underbrace{7}_{111} \bullet \underbrace{2}_{010})_8 = (\cancel{0}11111.01\cancel{0})_2$

## Hexa ⟷ Binary

$(1110010101.1011011)_2 \Rightarrow (?)_{16}$

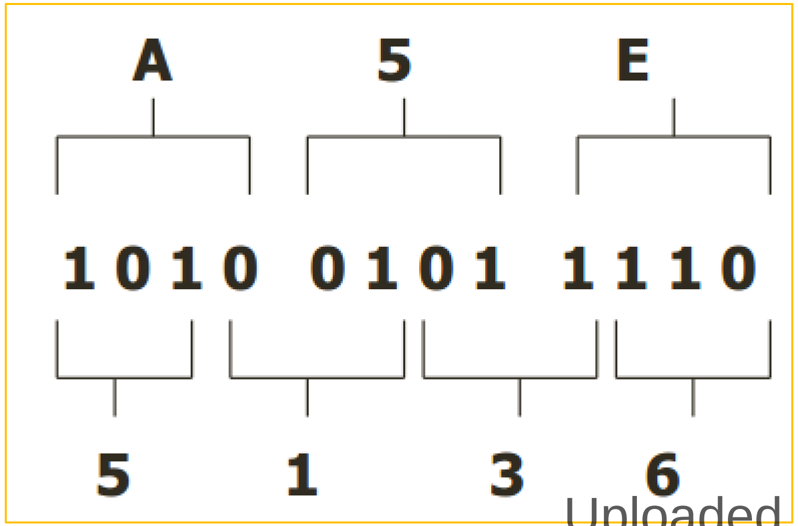
$(\underbrace{0011}_{3}\ \underbrace{1001}_{9}\ \underbrace{0101}_{5} \bullet \underbrace{1011}_{B}\ \underbrace{0110}_{6})_2 = (395.B6)_{16}$

$(37.2)_{16} \Rightarrow (?)_2$

$(\underbrace{3}_{0011}\ \underbrace{7}_{0111} \bullet \underbrace{2}_{0010})_{16} = (\cancel{00}110111.001\cancel{0})_2$

## Octal ⟷ Hexa

Binary as an intermediate step



## Conversion From Base r To Decimal:

- Expanding the number in a **power series** and adding all the terms as shown previously.

$$
\begin{aligned}
(1011.11)_2 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} \\
&= 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 + 1 \times 0.5 + 1 \times 0.25 \\
&= 8 + 2 + 1 + 0.5 + 0.25 \\
&= (11.75)_{10}
\end{aligned}
$$

## Conversion From Decimal To Base r:

- The number is separated into **integer** part and a **fraction**.
- The **integer** is successively **divided** by the **new base**, keeping track of the **remainder**, until the **quotient is zero**.
- The **fraction** is **multiplied** by the **new base**, keeping track of the **generated integer** part, until the required **accuracy** is reached.
- **Join** the **two** parts with the target radix **point**

ENCS 2340

## Conversion From Decimal to Binary:

- Divide the decimal number by **2**. Note down the **quotient** and the **remainder**.
- The remainder will be either 0 or 1. Label it as $\mathbf{a_0}$.
- If the quotient **is not 0**, divide it by 2 and note down the quotient and remainder.
- The remainder will be either 0 or 1. Label it as $\mathbf{a_1}$.
- **Repeat** the same process until the **quotient becomes 0**.

$(39)_{10} \rightarrow (????)_2$

| Divide by | Quotient | Remainder | Coefficient |
|---|---|---|---|
| 2 | 39 | | |
| 2 | 19 | 1 | $a_0 = 1$ |
| 2 | 9 | 1 | $a_1 = 1$ |
| 2 | 4 | 1 | $a_2 = 1$ |
| 2 | 2 | 0 | $a_3 = 0$ |
| 2 | 1 | 0 | $a_4 = 0$ |
| 2 | 0 | 1 | $a_5 = 1$ |

**$(39)_{10} \rightarrow (100111)_2$**

ENCS 2340

## More Examples:

$(53)_{10} \Rightarrow (?)_2$

| ÷ Radix | Quotient | Rem. |
|---|---|---|
| 53 ÷ 2 | 26 | 1 |
| 26 ÷ 2 | 13 | 0 |
| 13 ÷ 2 | 6 | 1 |
| 6 ÷ 2 | 3 | 0 |
| 3 ÷ 2 | 1 | 1 |
| 1 ÷ 2 | END←0 | 1 |

$(53)_{10} \Rightarrow (110101)_2$

$(51)_{10} \Rightarrow (?)_2$

| | | |
|---|---|---|
| 51 | • | ← Radix point |
| 25 | 1 | ← LSB |
| 12 | 1 | |
| 6 | 0 | |
| 3 | 0 | |
| 1 | 1 | |
| 0 | 1 | ← MSB |

$(51)_{10} \Rightarrow (110011.)_2$

# Decimal to Octal Conversion



- The conversion is just like the one we discussed on the Binary case.
- The only difference is that we now divide by **8** instead of 2.

$(39)_{10} \rightarrow (????)_8$

| Divide by | Quotient | Remainder | Coefficient |
|---|---|---|---|
| 8 | 39 | | |
| 8 | 4 | 7 | $a_0 = 7$ |
| 8 | 0 | 4 | $a_1 = 4$ |

$\mathbf{(39)_{10} \rightarrow (47)_8}$

**Example:**

$(755)_{10} \Rightarrow (?)_8$

| | |
|---|---|
| 755 | • |
| 94 | 3 |
| 11 | 6 |
| 1 | 3 |
| 0 | 1 |

$(755)_{10} \Rightarrow (1363.)_8$

# Decimal to Base r Conversion



- The conversion is just like the one we discussed on the Binary case.
- The only difference is that we now divide by **r** instead of 2.

$(1738)_{10} \rightarrow (????)_{16}$

| Divide by | Quotient | Remainder | Coefficient |
|---|---|---|---|
| 16 | 1738 | | |
| 16 | 108 | 10 | $a_0 = A$ |
| 16 | 6 | 12 | $a_1 = C$ |
| 16 | 0 | 6 | $a_2 = 6$ |

**$(1738)_{10} \rightarrow (6CA)_{16}$**

**Example (r =12):**

$(1606)_{10} \Rightarrow (?)_{12}$

| | |
|---|---|
| 1606 | • |
| 133 | 10=A |
| 11 | 1 |
| 0 | 11=B |

$(1606)_{10} \Rightarrow (B1A.)_{12}$

## Fractions Conversion



- Repeatedly **multiply** by the target base **r** and save the **integer** part of the result (always < r) until you get **0** fraction or **enough digits**
- The digits for the **new base** are those integers with the **first** being the **MSD**

$(0.6875)_{10} = (??)_2$

| Multiply By | Fraction | Result | Integer Part |
|---|---|---|---|
| 2 | 0.6875 | 1.375 | 1 |
| 2 | 0.375 | 0.75 | 0 |
| 2 | 0.75 | 1.5 | 1 |
| 2 | 0.5 | 1.0 | 1 |

$(0.6875)_{10} = (0.1011)_2$

- In the above example the fractional part **reached 0 exactly** as a result of the repeated multiplications → exact conversion was achieved**: Machine Number**
- In general it may take **many digits** in the target system to get this or it may **never** happen!
  Example: Convert $0.65_{10}$ to $()_2$ → $(0.65)_{10} = (0.1010011001 1001 \ldots)_2$
  The fractional part **repeating** every 4 steps, → 1001 repeats forever! → $0.10\overline{1001}$

**Solution:** Specify **required #** of digits to the **right** of radix point and **chop** or **round** to this number of bits

ENCS 2340

## More Examples:

$(0.731)_{10} \Rightarrow (?)_2$

r point$\rightarrow$ •

$0.731 \times 2 = \mathbf{1}.462$

$0.462 \times 2 = \mathbf{0}.924$

$0.924 \times 2 = \mathbf{1}.848$

$0.848 \times 2 = \mathbf{1}.696$

$0.696 \times 2 = \mathbf{1}.392$

$(0.731)_{10} \Rightarrow (0.10111)_2$

$(0.731)_{10} \Rightarrow (?)_8$

r point$\rightarrow$ •

$0.731 \times 8 = \mathbf{5}.848$

$0.848 \times 8 = \mathbf{6}.784$

$0.784 \times 8 = \mathbf{6}.272$

$0.272 \times 8 = \mathbf{2}.176$

$0.176 \times 8 = \mathbf{1}.408$

$(0.731)_{10} \Rightarrow (0.56621)_8$

**Chop @ 5 Digits**

$(0.357)_{10} \Rightarrow (?)_{12}$

r point$\rightarrow$ •

$0.357 \times 12 = \mathbf{4}.284$

$0.284 \times 12 = \mathbf{3}.408$

$0.408 \times 12 = \mathbf{4}.896$

$0.896 \times 12 = \mathbf{10}.752 \rightarrow A$

$(0.357)_{10} \Rightarrow (0.434A)_{12}$

**Chop @ 4 Digits**

## Comprehensive Conversion



Convert $(153.513)_{10}$ to Base **8**, **rounding** the resulting fraction to **3 octal** digits

### Integer Part: 153

| Divide By | Quotient | Remainder |
| --- | --- | --- |
| 8 | 153 | |
| 8 | 19 | 1 |
| 8 | 2 | 3 |
| 8 | 0 | 2 |

$\mathbf{(231)_8}$

### Fraction Part: 0.513

| Multiply By | Fraction | Result | Integer Part |
| --- | --- | --- | --- |
| 8 | 0.513 | 4.104 | 4 |
| 8 | 0.104 | 0.832 | 0 |
| 8 | 0.832 | 6.656 | 6 |
| 8 | 0.656 | 5.248 | 5 |

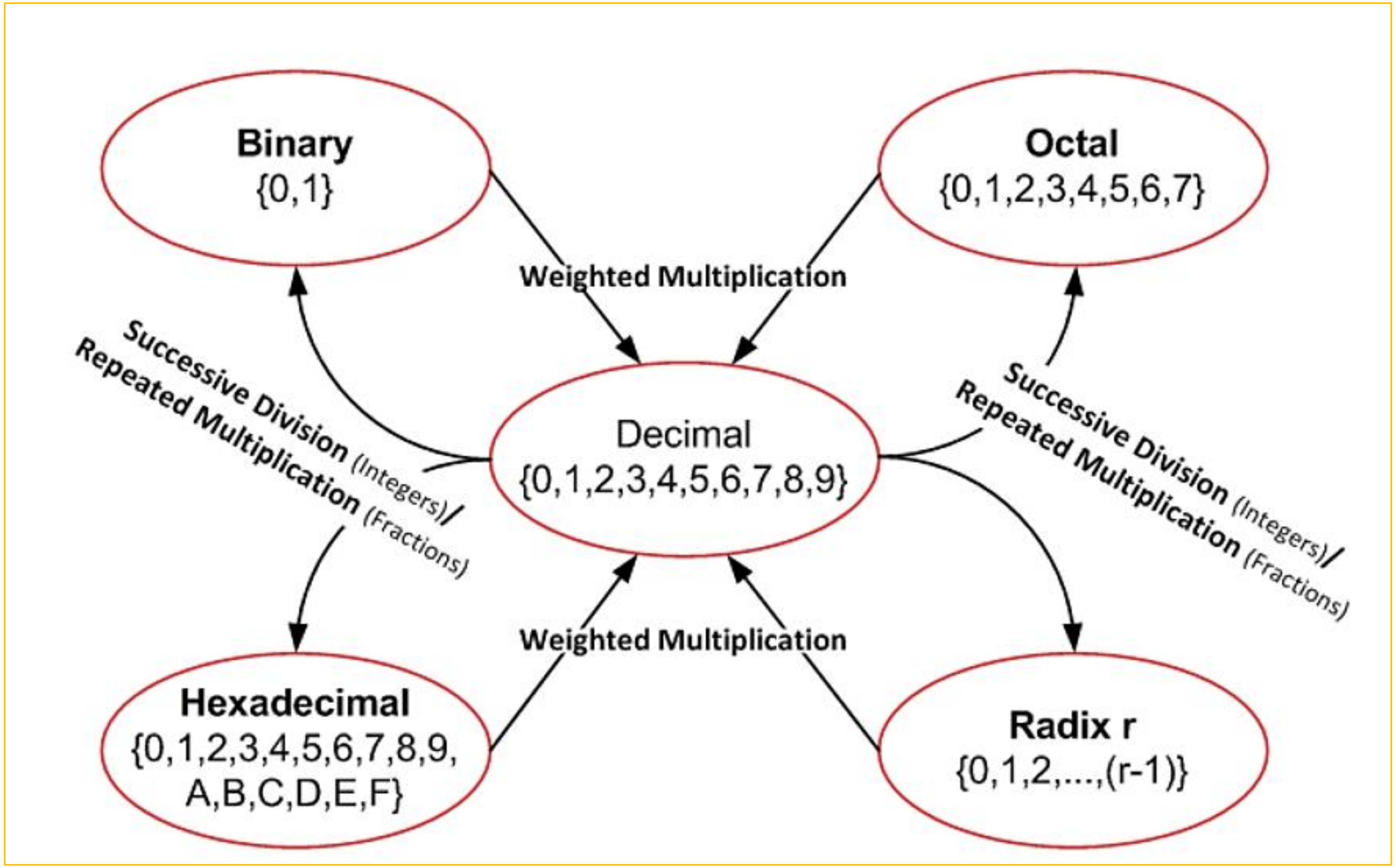
$\mathbf{(0.4065)_8}$

$(0.513)_{10} = (0.4065)_8 \rightarrow (0.407)_8$ after rounding, since 5>4 , we **add 1 to fraction**

$$\mathbf{(153.513)_{10} = (231.407)_8}$$

# Conversion Summary





- The rules used in decimal arithmetic operations are applied in any other number system.
  - Digit **carry** to the higher order position in **addition**.
  - Digit **borrow** from higher order position in **subtraction**.

## Binary Addition

**0 + 0 = 0**
**0 + 1 = 1**
**1 + 0 = 1**
**1 + 1 = 0** and carry 1 to the next column

Add $45_{10}$ and $44_{10}$ in binary

```
          1     1 1   <------------  Carries
45₁₀ =     1 0 1 1 0 1
44₁₀ =     1 0 1 1 0 0
         -------------
          1 0 1 1 0 0 1     = 89₁₀
```

## Binary Subtraction

**0 - 0 = 0**
**0 - 1 = 1** and borrow 1 from the next column
**1 - 0 = 1**
**1 - 1 = 0**

Subtract $108_{10}$ from $218_{10}$ in binary

```
          0 10 10 0 10 10  <------------  Borrows
218₁₀ =   1̸ 1̸ 0 1̸ 1̸ 0 1 0
108₁₀ =     1 1 0 1 1 0 0
         ------------------
            1 1 0 1 1 1 0 = 110₁₀
```

## Binary Multiplication

Multiply $110110_2$ times $101101_2$ in binary

```
      1 1 0 1 1 0        54
    x 1 0 1 1 0 1      x 45
```

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | | | | | |
| | | | | | | 1 | 1 | 0 | 1 | 1 | 0 |
| | | | | | 0 | 0 | 0 | 0 | 0 | 0 | |
| | | | | 1 | 1 | 0 | 1 | 1 | 0 | | |
| | | | 1 | 1 | 0 | 1 | 1 | 0 | | | |
| | | 0 | 0 | 0 | 0 | 0 | 0 | | | | |
| | 1 | 1 | 0 | 1 | 1 | 0 | | | | | |
| **1** | **0** | **0** | **1** | **0** | **1** | **1** | **1** | **1** | **1** | **1** | **0** |

= 2430

- **Complements** are used to simplify **subtraction** operation in digital computers

- Simplification of operation has multiple advantages:
  - It results in simpler circuits (convenience in designing process)
  - It results in low cost (simpler circuit means fewer and simpler hardware components)

- For each base (r), there are **two** complements:
  1) **Diminished** Radix**:** **(r-1)'s** Complement
  2) **Radix**: **(r's)** Complement

| Binary | Decimal | Octal |
|---|---|---|
| 1's | 9's | 7's |
| 2's | 10's | 8's |

- For a number **N** that is represented by **n** digits in radix r:
  - (r-1)'s complement of N is $[(r^n - 1) - N]$
  - r's complement of N is $[r^n - N]$ → (r-1)'s complement **+1**

| n=3 | | | |
|---|---|---|---|
| | Binary | Decimal | Octal |
| (r-1)'s | 7 - N | 999 - N | 511 - N |
| r's | 8 - N | 1000 - N | 512 - N |

- **9's** complement of the decimal number **N** = $(10^n - 1) - N$

**Subtract each digit from 9**

**Example:** Find the 9's comp. of $(134795)_{10}$ **[n=6]**

$= (10^6 - 1) - (134795)$
$= (999999) - (134795)$
$= (865204)_{10}$

- **10's** complement of the decimal number **N** = $(10^n - N)$

**9's complement + 1**

**Quick Method:** To find the 10's complement of a decimal number

1) Leave all **least-significant** zeros **unchanged** (Rightmost Zeros)
2) **Subtract** the **first** non-zero digit from **10** and all the **remaining** digits from **9's**

**Example:** Find the 10's comp. of $(134795)_{10}$ **[n=6]**

$= (10^6 - 134795)$
$= (1000000 - 134795)$
$= (865205)_{10}$

**= (865204) +1**



Mohammed Khalil

- **1's** complement of the decimal number **N** = $(2^n - 1) - N$

**Subtract each digit from 1**

**Two Possibilities:**

1) If the bit is 0 then its 1's complement is 1 − 0 = 1
2) If the bit is 1 then its 1's complement is 1 − 1 = 0

**Simply Flip each bit**

- **2's** complement of the decimal number **N** = $(2^n - N)$

**1's complement + 1**

**Quick Method:** To find the 2's complement of a decimal number

1) Leave all **least-significant** **0's** and the **first** **1** **unchanged** (Rightmost Zeros)
2) **Flip** all the **remaining** bits

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

**N = Complement(Complement(N))**

## Example:

Consider a binary number: 1011001
The 1's complement is: 0100110
The 2's complement is: 0110111

Consider a binary number: 00100100
The 1's complement is: 11011011
The 2's complement is
A) (11011011) + 1 = (11011100)

B) 0 0 1 0 0 1 0 0

$\Rightarrow$ 1 1 0 1 1 1 0 0

Flip | Same

# Subtraction using Complements



- Assume, we need to subtract Y from X → (X − Y)
  - There are 2 Cases: 1) X ≥ Y  2) X < Y
  - We can Use 1's or 2's to perform the subtraction

## Case 1: X ≥ Y

**Using 1's**

1) Add X and the **1's complement** of Y
2) If an **end carry** occurs, we **add 1** to the result to get the final answer

Consider $X = 11101$ and $Y = 10111$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $X$ | $=$ | | | 1 | 1 | 1 | 0 | 1 |
| 1's complement of $Y$ | $=$ | $+$ | | 0 | 1 | 0 | 0 | 0 |
| Sum | $=$ | | 1 | 0 | 0 | 1 | 0 | 1 |
| End carry | $=$ | $+$ | | | | | | 1 |
| $X - Y$ | $=$ | | | 0 | 0 | 1 | 1 | 0 |

end carry

**Using 2's**

1) Add X and the **2's complement** of Y
2) If an **end carry** occurs, **discard** it to get the final answer

Consider $X = 11101$ and $Y = 10111$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $X$ | $=$ | | | 1 | 1 | 1 | 0 | 1 |
| 2's complement of $Y$ | $=$ | $+$ | | 0 | 1 | 0 | 0 | 1 |
| Sum | $=$ | | 1 | 0 | 0 | 1 | 1 | 0 |
| Discard End carry | $=$ | | ~~1~~ | | | | | |
| $X - Y$ | $=$ | | | 0 | 0 | 1 | 1 | 0 |

end carry

## Case 2: X < Y

- When (X < Y) → (X – Y) will be a **negative** number.
- Therefore, the result we will get using the previous method will be the 1's or 2's complement.
- So, to get the final result:
  1) Find the complement once again
  2) Append a negative sign to it

**Using 1's**

Consider $X = 10111$ and $Y = 11101$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $X$ | $=$ | | 1 | 0 | 1 | 1 | 1 |
| 1's complement of $Y$ | $=$ | $+$ | 0 | 0 | 0 | 1 | 0 |
| Sum | $=$ | | 1 | 1 | 0 | 0 | 1 |
| 1's complement of the sum | $=$ | | 0 | 0 | 1 | 1 | 0 |
| $X - Y$ | $=$ | – | 0 | 0 | 1 | 1 | 0 |

**Using 2's**

Consider $X = 10111$ and $Y = 11101$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $X$ | $=$ | | 1 | 0 | 1 | 1 | 1 |
| 2's complement of $Y$ | $=$ | $+$ | 0 | 0 | 0 | 1 | 1 |
| Sum | $=$ | | 1 | 1 | 0 | 1 | 0 |
| 2's complement of the sum | $=$ | | 0 | 0 | 1 | 1 | 0 |
| $X - Y$ | $=$ | – | 0 | 0 | 1 | 1 | 0 |

**Remember:** No end carry is generated → The answer is **negative**

ENCS
2340

## What About Decimal Numbers?

### Case 1: X ≥ Y

Subtract (76425 – 28321) using 9's complements.

The 9's complement of 28321 is 71678.

```
                    76425
                  + 71678
End carry →       1|48103
                        1
                    48104
```

Subtract (76425 – 28321) using 10's complements.

The 10's complement of 28321 is 71679.

```
   76425
 + 71679
  148104   (carry 1 discarded)
```

### Case 2: X < Y

Subtract (285.31 – 3459.20) using 9's complements.

The 9's complement of of 3459.20 is 6540.79.

```
                    285.31
                 + 6540.79
No end carry →     6826.10
```

Therefore the difference is negative and is equal to the 9's complement of the answer, - (6826.10)' = - 3173.89

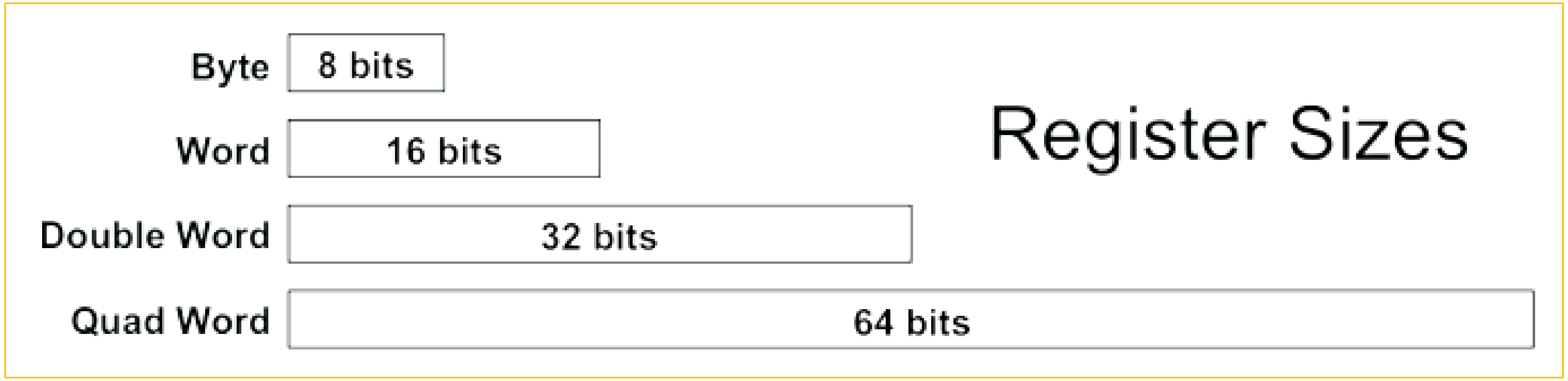
Subtract (28531 – 345920) using 10's complements.

The 10's complement of of 345920 is 654080.

```
                     28531
                  + 654080
No end carry →      682611
```

Therefore the difference is negative and is equal to the 10's complement of the answer, - (682611)' = - 317389

- Digital computers store numbers in special digital electronic devices called **Registers**.
- Registers consist of an **n** fixed number of storage elements that is typically a power of **2** (Bits).
- An **n-bit** register can store maximum of $\mathbf{2^n}$ Distinct Values.
- Values stored in registers may be either **unsigned** or **signed** numbers.



## Unsigned Numbers

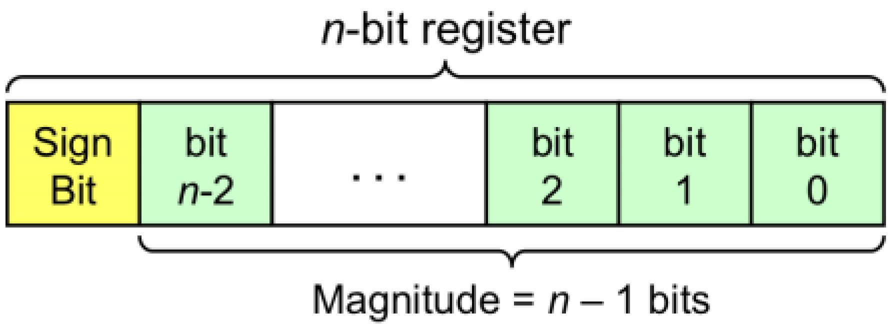
- An n-bit register can store any **unsigned** number that has **n-bits** or less.
- When representing an **integer** number, this n-bit register can hold values from **0** up to **$(2^n-1)$.**
- No **sign** information needs to be represented.

## Signed Numbers

- An n bits of the register should represent the **magnitude** of the number and its **sign** as well.
- Two major techniques are used to represent **signed** numbers:
  1) Signed Magnitude Representation.
  2) Complement method:
     - 1's Complement.
     - 2's Complement

# Signed Magnitude Number Representation






- Independent representation of the **sign** and **magnitude**.
- Leftmost bit is the sign bit: **0** is positive and **1** is negative.
- Using n bits, **largest** represented magnitude = $\mathbf{2^{(n-1)} - 1}$
- Symmetric range of represented values for n-bit register; from $-(2^{(n-1)} - 1)$ to $+(2^{(n-1)} - 1)$.
- For 8-bit register (n=8) → -127 to +127

## Examples:

- ✓ X = 1001 (-1, 4-bit)
- ✓ X = 0110 (+6, 4-bit)
- ✓ X = 00001111 (+15, 8-bit)
- ✓ X = 10000011 (-3, 8-bit)
- ✓ X = 11001111 (-79, 8-bit)
- ✓ X = 110111 (-23, 6-bit)

## 5-bit signed-magnitude binary numbers

| | |
|---|---|
| +15 | 01111 |
| +14 | 01110 |
| +13 | 01101 |
| +12 | 01100 |
| +11 | 01011 |
| +10 | 01010 |
| +9 | 01001 |
| +8 | 01000 |
| +7 | 00111 |
| +6 | 00110 |
| +5 | 00101 |
| +4 | 00100 |
| +3 | 00011 |
| +2 | 00010 |
| +1 | 00001 |
| +0 | 00000 |

| | |
|---|---|
| -0 | 10000 |
| -1 | 10001 |
| -2 | 10010 |
| -3 | 10011 |
| -4 | 10100 |
| -5 | 10101 |
| -6 | 10110 |
| -7 | 10111 |
| -8 | 11000 |
| -9 | 11001 |
| -10 | 11010 |
| -11 | 11011 |
| -12 | 11100 |
| -13 | 11101 |
| -14 | 11110 |
| -15 | 11111 |

# Signed Complement Number Representation



- 1's and 2's complement can also be used to represent signed numbers
- Since usage of 2's complement simplifies computations, we almost always use it to represent negative numbers
- 2's Complement
  - Only **one** representations for zero.
  - **Asymmetric** range of represented values for n-bit register;
    From $-(2^n-1)$ to $+(2^{n-1}-1)$.
    For 8-bit register (n=8) → -128 to +127

| Decimal | Signed Magnitude | Signed 2's Complement | Signed 1's Complement |
|---|---|---|---|
| +7 | 0111 | 0111 | 0111 |
| +6 | 0110 | 0110 | 0110 |
| +5 | 0101 | 0101 | 0101 |
| +4 | 0100 | 0100 | 0100 |
| +3 | 0011 | 0011 | 0011 |
| +2 | 0010 | 0010 | 0010 |
| +1 | 0001 | 0001 | 0001 |
| +0 | 0000 | 0000 | 0000 |
| −0 | 1000 | — | 1111 |
| −1 | 1001 | 1111 | 1110 |
| −2 | 1010 | 1110 | 1101 |
| −3 | 1011 | 1101 | 1100 |
| −4 | 1100 | 1100 | 1011 |
| −5 | 1101 | 1011 | 1010 |
| −6 | 1110 | 1010 | 1001 |
| −7 | 1111 | 1001 | 1000 |
| −8 | — | 1000 | — |

## Examples:

| Binary | Unsigned | Signed-magnitude | 1's complement signed | 2's complement signed |
|---|---|---|---|---|
| 10111000 | 184 | -56 | -71 | -72 |
| 01101001 | 105 | +105 | +105 | +105 |
| 10000111 | 135 | -7 | -120 | -121 |
| 11000111 | 199 | -71 | -56 | -57 |

# Sign Extension

- Before we can add / subtract two **signed** numbers, they must have the **same** number of bits
- Also we may need to move a **signed** number from a small register to a larger register

| - 5 | in 4 bits | in 8 bits | |
|---|---|---|---|
| Signed-Magnitude | 1101 | 10000101 | (sign-magnitude) |
| Signed 1's Complement | 1010 | 11111010 | (sign extension) |
| Signed 2's Complement | 1011 | 11111011 | (sign extension) |

**Examples:**

$(10110011)_2$ is in 2's complement, put it in a 16 bits.

$(10110011)_2 = -77 \Rightarrow (11111111\ \underline{1}0110011)_2 = -77$

$(01100010)_2$ is in 2's complement, but it in a 16 bits.

$(01100010)_2 = +98 \Rightarrow (00000000\ \underline{0}1100010)_2 = +98$

- In signed magnitude representation:
  1) If the signs are the **same**, **add** the magnitudes and give the sum the same sign.
  2) If **different** signs, **subtract** and give the result the sign of the big number.

- In complement representation:
  1) Add the two numbers including the sign bit.
  2) Any carry out from the sign bit is **ignored**.
  3) No comparison or subtraction is needed.

**Examples:**

Add (-6) + (+13) using signed 2's complement form with 8 bits. Repeat for (+6) + (-13).

**Examples:**

Add (-6) + (+13) using signed 2's complement form with 8 bits. Repeat for (+6) + (-13).

$(+6) \equiv 00000110$ and $(+13) \equiv 00001101$
$(-6\ \ ) \equiv 11111010$ and $(-13\ ) \equiv 11110011$

$$\begin{array}{rrl} +6 \rightarrow & 00000110 & \\ -13 \rightarrow & \underline{11110011} & \\ & 11111001 & \rightarrow -7 \end{array}$$

$$\begin{array}{rrl} -6 \rightarrow & 11111010 & \\ +13 \rightarrow & \underline{00001101} & \\ & \not{1}|00000111 & \rightarrow +7 \end{array}$$

# Carry and Overflow



- **Carry** is important when adding/subtracting **unsigned** integers to indicates that the unsigned sum is **out of range**. (SUM < 0 or SUM > maximum unsigned n-bit value).
- **Overflow** is important when adding/subtracting **signed** integers to indicates that the signed sum is **out of range**.
- Overflow occurs when:
  1) Adding two **positive** numbers and the sum is **negative**.
  2) Adding two **negative** numbers and the sum is **positive**.
- **We can have carry without overflow and vice-versa**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| carries | | | | | 1 | | | | |
| | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 15 |
| + | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 8 |
| | | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 23 |

Carry = 0 Overflow = 0

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| carries | 1 | 1 | 1 | 1 | 1 | | | | | |
| | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 15 |
| + | | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 248 (-8) |
| | | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 7 |

Carry = 1 Overflow = 0

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| carries | | 1 | | | | | | | | |
| | | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 79 |
| + | | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 64 |
| | | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 143 (-113) |

Carry = 0 Overflow = 1

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| carries | 1 | | | 1 | 1 | | | | | |
| | | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 218 (-38) |
| + | | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 157 (-99) |
| | | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 119 |

Carry = 1 Overflow = 1

**Indicators of the occurrence of Overflow:**

1) The two added binary numbers have the same sign and the result has the opposite sign

2) Carries from the last 2 bits in the binary addition operation are different (1,0 or 0,1)

3) Also the result will be wrong

- Digital systems and circuits can only store one of **two** states,"0" and "1".
- **One** bit can represent **two** elements only!
- With **n** bits, we can produce $\mathbf{2^n}$ different combinations.
- To represent **m** elements, we need **n** bits, where $\mathbf{2^n \geq m}$.
- n = **Ceiling**($\log_2 m$)

- To code the decimal digits [0-9], we need at least **four** bits [$\log_2 10 = 3.322$]
- We call this binary-coded decimal (**BCD**)

# BCD



- **BCD** is a way to express each **decimal** digit with a binary code
- It is very easy to convert between **decimal** and **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 |

- Four bits can be used to represent 16 numbers
- In BCD we utilize only **10**. The remaining **6** code combinations are not used – **invalid codes**
- The invalid codes are: 1010, 1011, 1100, 1101, 1110, 1111

## How to Express a Decimal Number in BCD??

**Simply replace each decimal digit with the appropriate 4-bit code**

$$\overbrace{1000}^{8}\,\overbrace{0010}^{2}$$

$$\overbrace{0001}^{1}\,\overbrace{0000}^{0}\,\overbrace{0110}^{6}$$

$$\overbrace{1001}^{9}\,\overbrace{0000}^{0}\,\overbrace{0001}^{1}\,\overbrace{0010}^{2}$$

## How to Determine a Decimal Number From a BCD??

**1) Start from the right-most bit and break the code into group of four bits**
**2) Write the decimal digit represented by each 4-bits group**

$$\underbrace{1001}_{9}\,\underbrace{1001}_{9}$$

$$\underbrace{0001}_{1}\,\underbrace{0011}_{3}\,\underbrace{1001}_{9}$$

# BCD Addition



- The addition of two BCD digits with a possible carry from the previous less significant pair of digits results in a sum in the range (**0 to 19**). There is a difference in the representation of the sum in binary and in BCD code.
- If 0 ≤ **Sum** ≤ 9 → sum in BCD = sum in binary. (Done)
- If 10 ≤ **Sum** ≤ 19 then, sum in BCD consists of **8 bits** which is not equal to the sum in binary. Corrected by **adding 0110** to the binary sum.

**Examples:**

| | | | | |
|---|---|---|---|---|
| | 0001 | 0011 | | 13 |
| + | 0010 | 0110 | + | 26 |
| | 0011 | 1001 | | 39 |

| | | | |
|---|---|---|---|
| 4 → | 0100 | 4 → | 0100 |
| +5 → | 0101 | +8 → | 1000 |
| 9 → | 1001 | 12 → | 1100 |
| | | | +0110 |
| | | | 0001 0010 |

ENCS 2340

## More Examples:

| | | | | |
|---|---|---|---|---|
| | 1001 | | | 9 |
| + | 1001 | | + | 9 |
| 0001 | 0010 | Invalid because of carry | | 18 |
| + | 0110 | Add 6 | | |
| 0001 (1) | 1000 (8) | Valid BCD number | | |

Solve this in BCD : (+375)+(-240).

the 10's complement of (-)240 is (9)760.

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 11 11 11 |
| 0 | 375 | 0000 | 0011 0111 0101 |
| +9 | 760 | +1001 | 0111 0110 0000 |
| ~~1~~0 | 135 | 1010 | 1011 1101 0101 |
| | | 0110 | 0110 0110 ↓ |
| | | ~~1~~0000 | 0001 0011 0101 |

| | | | | | |
|---|---|---|---|---|---|
| | 0001 | 0110 | | | 16 |
| + | 0001 | 0101 | | + | 15 |
| | 0010 | 1011 | Right group is invalid ($> 9$), left is valid | | 31 |
| + | 0001 | 0110 | Add 6 to invalid code, add carry to next group | | |
| | 0011 (3) | 0001 (1) | Valid BCD number | | |

# Other Decimal Codes



- 8421, 2421, and (8,4,-2,-1) are **weighted** codes
- Excess-3 is an **unweighted code**
- 2421 and Excess-3 codes exhibit self-complementing property
- **Self-complementing:** 9's complement of a decimal number is obtained directly by **flipping** the bits of the code of that decimal number

| 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 | 1101 | 1010 | 1001 |
| 8 | 1000 | 1110 | 1011 | 1000 |
| 9 | 1001 | 1111 | 1100 | 1111 |
| | 1010 | 0101 | 0000 | 0001 |
| Unused bit combi- nations | 1011 | 0110 | 0001 | 0010 |
| | 1100 | 0111 | 0010 | 0011 |
| | 1101 | 1000 | 1101 | 1100 |
| | 1110 | 1001 | 1110 | 1101 |
| | 1111 | 1010 | 1111 | 1110 |

- Using Excess-3 code the decimal number **428** is represented as: **0111 0101 1011**
- 9's complement of 428 is: 999-428 = **571**
- 9's complement of 428 in encoded form (Excess-3 form) is: **1000 1010 0100** (flip each bit)
- This corresponds to the 1's complement of a binary number

# *Gray Code*



- **Unweighted** code, Non-numeric code
- Only a **single** bit **changes** from one code word to the next in sequence
- **No** two codes are **identical**, Gray codes can have any number of bits

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

# *ASCII Character Code*



- Standard code for alphanumeric characters
- 7-bit code (a byte is usually used, additional codes used to represent Greek or italic type fonts)

| | $b_7b_6b_5$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $b_4b_3b_2b_1$ | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| 0000 | NUL | 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 |

# Error-Detecting Code



- A parity bit is added to the ASCII codes to **detect** errors
- A parity bit is an extra bit to make the total number of 1's either **even** or **odd**
- The code for A is 1000001 → number of 1's is 2
- Insert a parity bit on the left side to make the total number of 1's either even or odd
  - The code for A with even parity is 01000001
  - The code for A with odd parity is 11000001

## Example:

**75.8**

Considering the number above (including the decimal point) as **four** characters.
Represent each character as a **7-bit ASCII** binary codes **+** an additional **odd parity bit** appended as the **MSB** (i.e. to the left of the ASCII code).
Express the number as a sequence of 8 hexadecimal digits.

| **75.8** | | | |
|---|---|---|---|
| | 7-bit ASCII | + Parity bit | Hex |
| 7 | 0110111 | 00110111 | 37 |
| 5 | 0110101 | 10110101 | B5 |
| . | 0101110 | 10101110 | AE |
| 8 | 0111000 | 00111000 | 38 |

**37 B5 AE 38**

- Binary **logic** deals with binary quantities which can take one of two values (0 & 1, True & False, ... etc).
- A binary number can be represented by a **variable** (x, y, z, A, B, C ... etc).
- Binary **variables** take on one of **two** values.
- Logical **operators** operate on binary values and binary variables
- Three basic logical operations; AND (.), OR (+), NOT ( ' ).

**Truth Table**

| AND | | | OR | | | NOT | |
|---|---|---|---|---|---|---|---|
| $x$ | $y$ | $x \cdot y$ | $x$ | $y$ | $x + y$ | $x$ | $x'$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | | |
| 1 | 1 | 1 | 1 | 1 | 1 | | |

## Logic Gates

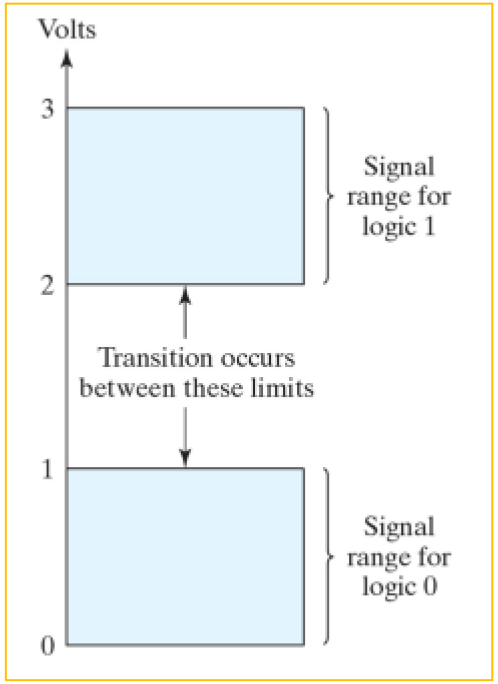


- Logic gates are electronic circuits that operate on one or more physical input signals to produce an output signal

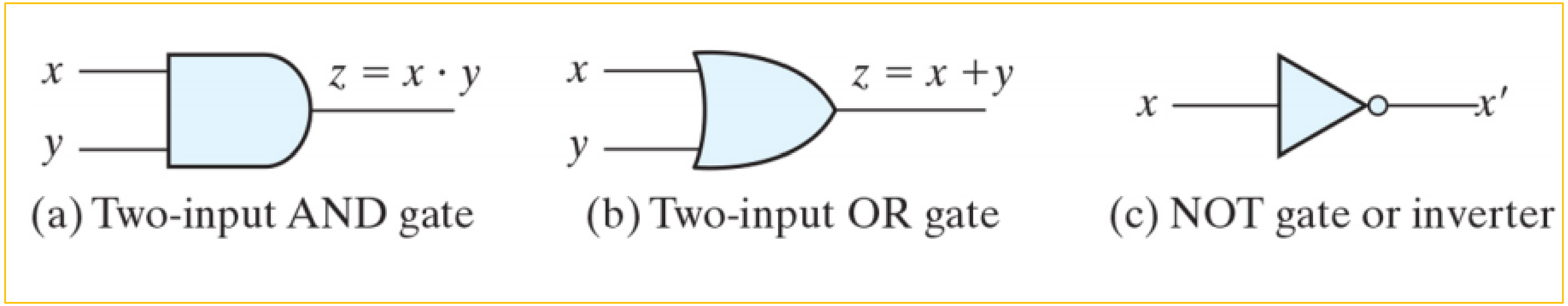
- These input signals could be current or voltage signals.
  - For example: voltage signals varying from 0 V to 3 V

- Digital systems work on two states only.
  - Therefore, we may assign 0 V as a 0 state (OFF) and assign 3 V as a 1 state (ON)

- In practice, 0 (OFF) will have a range of 0-1.5 V and 1 (ON) will have a range of 1.5-3 V



**Logic Gates**




**Signal Waveform**