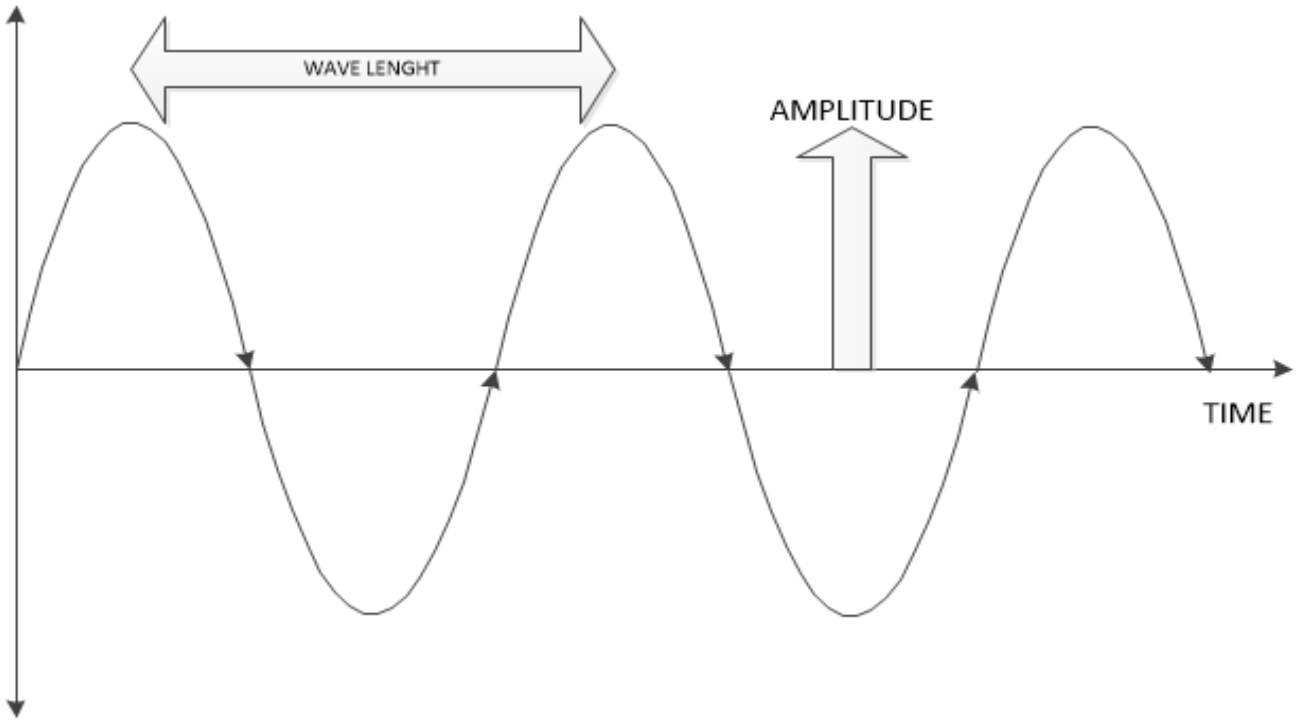
INDEX

1)SIGNALS

1.1) ANALOG SIGNAL

Analog signal is a signal continuous in time i.e. it varies according to the time which in simple words is that when going from one point to other let us say from point A to point B it will include infinite values along it travelling path.

For e.g.



Analog signal has a property that it uses the property of the medium to carry the message for example Pressure of the sound wave continuously varies with the voltage .
An analog signal can be depicted using three properties :-

- **Amplitude**: - Amplitude is a signal's strength and can be visualised as the height of the signal. Therefore, the higher the amplitude the "louder" the signal. Signal strength is typically measured in decibels, named after the inventor of the telephone: Alexander Graham Bell.

- **Phase**: - Phase is the rate at which a signal changes its own relationship to zero time, and is expressed in radians or degrees. One complete cycle of a wave begins at a predetermined point and continues until the same point is reached again. The phase shifts forward or backward along the time axis. The amount of shift can be from 0 to 360 degrees. Therefore, a shift of 360 degrees is one complete period.

- **Frequency**: - Frequency is the rate at which a signal changes per second and is typically measured in hertz (Hz). Therefore, if a signal has a frequency of 100 Hz, it changes at a rate of 100 times per second. This rate of change is also referred to as the number of cycles per second. Frequency can also be measured in kilohertz (kHz), megahertz (MHz), and gigahertz (GHz).

1.1.1) Advantages of Analog Signal

Music and speech as well as most other sound is analog. We interpret analog audio signals almost instantly, and without even thinking about it. Without analog signals, most of our listening activities are for nothing.

- One of the major advantages of the analog signal is that they have power to define infinite amount of data.
- Density of the analog signals is much higher as compared to digital ones.
- Analog signals have easy processing.

1.1.2) Disadvantages of Analog Signal

- **Losses** in Analog Signal
- **Noise** In Analog Signal The primary disadvantage of analog signalling is that any system has
  Noise -random unwanted variation. As the signal is copied and re-copied, or transmitted over long distances, these apparently random variations become dominant.
  The effects of noise create signal loss and distortion. This is impossible to recover
- **Distortion** in Analog Signal - It is defined to be the change of shape in signal getting transmitted from one point to the other or from the source to the destination Induced Noise
- **Thermal Noise**- Noise due to random motion of electrons cause this type of noise in the signal and the shape of signal gets distorted because of this

- **Attenuation** - It is loss of energy in the signal as it is transmitted from the source in destination. This mostly occurs in case the signal has to be transferred over a long distance to overcome this effect we use amplifiers which amplify the signal. But there is a problem in this case when we amplify the given signal the noise also gets amplified if the noise is induced in the signal

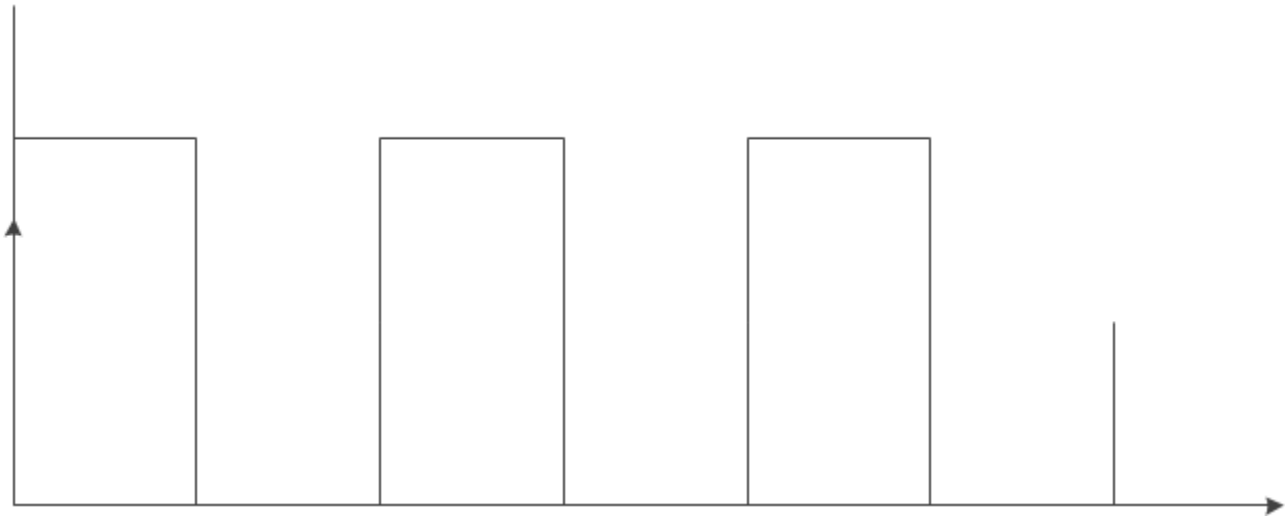
  For instance, sunlight is attenuated by dark glasses

1.2) DIGITAL SIGNAL

A digital signal is a signal that represents a sequence of discrete values. A logic signal is a digital signal with only two possible values, and describes an arbitrary bit stream. Other types of digital signals can represent three-valued logic or higher valued logics.

With digital signals, system noise, provided it is not too great, will not affect system operation whereas noise always degrades the operation of analog signals to some degree.

For e.g.



1.2.1) Disadvantages of Digital Signal

- **Sampling Error** - Digital communications require greater bandwidth than analogue to transmit the same information. The detection of digital signals requires the communications system to be synchronised, whereas generally speaking this is not the case with analogue systems.
- Samples analog wave forms into a **limited set of numbers** and records them. Hence some information is lost. Less accuracy.
- **Cost** is high and not easily portable
- **Quantization** errors, high bandwidth requirements due to pulse signals, less accurate due to finite set of data.

1.2.2) Advantages of Digital Signal

Fast processing, easier for storage, strong immunity to noise, parallel processing possibility, error correction possibilities, easy portability.

1. The main advantage of digital signals over analog signals is that the precise signal level of the digital signal is not vital. This means that digital signals are fairly immune to the imperfections of real electronic systems which tend to spoil analog signals. As a result, digital CD's are much more robust than analog LP's.
2. Codes are often used in the transmission of information. These codes can be used either as a means of keeping the information secret or as a means of breaking the information into pieces that are manageable by the technology used to transmit the code, e.g. The letters and numbers to be sent by a Morse code are coded into dots and dashes.
3. Digital signals can convey information with greater noise immunity, because each information component (byte etc.) is determined by the presence or absence of a data bit (0 or one). Analog signals vary continuously and their value is affected by all levels of noise.
4. Digital signals can be processed by digital circuit components, which are cheap and easily produced in many components on a single chip. Again, noise propagation through the demodulation system is minimized with digital techniques.
5. Digital signals do not get corrupted by noise etc. You are sending a series of numbers that represent the signal of interest (i.e. audio, video etc.)
6. Digital signals typically use less bandwidth. This is just another way to say you can cram more information (audio, video) into the same space.
7. Digital can be encrypted so that only the intended receiver can decode it (like pay per view video, secure telephone etc.)
8. Enables transmission of signals over a long distance.
9. Transmission is at a higher rate and with a wider broadband width.
10. It is more secure.
11. It is also easier to translate human audio and video signals and other messages into machine language.
12. There is minimal electromagnetic interference in digital technology.
13. It enables multi-directional transmission simultaneously.

2) Digitization

Digitizing or digitization [1] is the representation of an object, image, sound, document or signal (usually an analog signal) by generating a series of numbers that describe a discrete set of its points or samples. The result is called digital representation or, more specifically, a digital image, for the object, and digital form, for the signal. In modern practice, the digitized data is in the form of binary numbers, which facilitate computer processing and other operations, but strictly speaking, digitizing simply means the conversion of analog source material into a numerical format; the decimal or any other number system can be used instead.

Digitization is of crucial importance to data processing, storage and transmission, because it "allows information of all kinds in all formats to be carried with the same efficiency and also intermingled". [2] Unlike analog data, which typically suffers some loss of quality each time it is copied or transmitted, digital data can, in theory, be propagated indefinitely with absolutely no degradation. This is why it is a favoured way of preserving information for many organisations around the world.

2.1) Process of Digitization

The term digitization is often used when diverse forms of information, such as text, sound, image or voice, are converted into a single binary code. Digital information exists as one of two digits, either 0 or 1. These are known as bits (a contraction of binary digits) and the sequences of 0s and 1s that constitute information are called bytes.

Analog signals are continuously variable, both in the number of possible values of the signal at a given time, as well as in the number of points in the signal in a given period of time. However, digital signals are discrete in both of those respects – generally a finite sequence of integers – therefore a digitization can, in practical terms, only ever be an approximation of the signal it represents.

Digitization occurs in two parts:

2.1.1) Discretization

The reading of an analog signal A, and, at regular time intervals (frequency), sampling the value of the signal at the point. Each such reading is called a sample and may be considered to have infinite precision at this stage;

2.1.2) Quantization

Samples are rounded to a fixed set of numbers (such as integers), a process known as quantization.

In general, these can occur at the same time, though they are conceptually distinct.

A series of digital integers can be transformed into an analog output that approximates the original analog signal. Such a transformation is called a DA conversion. The sampling rate and the number of bits used to represent the integers combine to determine how close such an approximation to the analog signal a digitization will be.

3)ANALOG SIGNAL TO DIGITAL SIGNAL

Analog signals are continuous electrical signals; digital signals are non-continuous. Analog signal can be converted to digital signal by ADC. [7]

Nearly all recorded music has been digitized. About 12 percent of the 500,000+ movies listed on the Internet Movie Database are digitized on DVD. [citation needed]

Handling of analog signal becomes easy [according to whom?] when it is digitized because the signal is digitized before modulation and transmission. The conversion process of analog to digital consists of two processes: sampling and quantizing.

Digitization of personal multimedia such as home movies, slides, and photographs is a popular method of preserving and sharing older repositories. Slides and photographs may be scanned using an image scanner, but videos are more difficult. [8]

4)ENCODING

4.1) Unary encoding

Unary coding, sometimes called **thermometer code**, is an entropy encoding that represents a natural number, *n*, with *n* ones followed by a zero (if *natural number* is understood as *non-negative integer*) or with *n* − 1 ones followed by a zero (if *natural number* is understood as *strictly positive integer*). For example, 5 is represented as 111110 or 11110. Some representations use *n* or *n* − 1 zeros followed by a one. The ones and zeros are interchangeable without loss of generality. Unary coding is both a Prefix-free code and a Self-synchronizing code.

n (non-negative)	n (strictly positive)	Unary code	Alternative
0	1	0	1
1	2	10	01
2	3	110	001
3	4	1110	0001
4	5	11110	00001
5	6	111110	000001
6	7	1111110	0000001
7	8	11111110	00000001
8	9	111111110	000000001
9	10	1111111110	0000000001

Unary code in use today

Examples of unary code uses include:

- In Golomb Rice code, unary encoding is used to encode the quotient part of the Golomb code word.

- In UTF-8, unary encoding is used in the leading byte of a multi-byte sequence to indicate the number of bytes in the sequence, so that the length of the sequence can be determined without examining the continuation bytes.
- Instantaneously trained neural networks use unary coding for efficient data representation.

Unary coding in biological networks

New research has shown that unary coding is used in the neural circuits responsible for birdsong production.[1][2] The nucleus in the brain of the songbirds that plays a part in both the learning and the production of bird song is the HVC (high vocal centre). This coding works as space coding which is an efficient strategy for biological circuits due to its inherent simplicity and robustness.

4.2) BINARY ENCODING

When we insert any character or symbol to a digital system, through key board, it is needed to be encoded in machine readable farm. Digital systems like computer etc., cannot read the characters or symbol directly. The system reads and computes any characters, numbers and symbols in their digital form. An encoder does the job that means, it converts different human readable characters or symbol to their equivalent digital format. An encoder is basically multi inputs and multi outputs digital logic circuit, which has as many inputs as the number of character to be encoded and as many outputs as the number of bits in encoded form of characters. Suppose we have to design an encoder which will encode 10 characters (from 0 to 9). The encoded form of each character would be 4-bit binary equivalent. Then the encoder will have 10 numbers of input lines and each for one character. There will be four output lines to represent 4-bit encoded form of each input character.

Similarly, for encoding M numbers of characters in N bit format, we need M Input N output digital encoder.

4.3) Variable-Length encoding

In coding theory a **variable-length code** is a code which maps source symbols to a *variable* number of bits.

Variable-length codes can allow sources to be compressed and decompressed with *zero* error (lossless data compression) and still be read back symbol by symbol. With the right coding strategy an independent and identically-distributed source may be compressed almost arbitrarily close to its entropy. This is in contrast to fixed length coding methods, for which data compression is only possible for large blocks of data, and any compression beyond the logarithm of the total number of possibilities comes with a finite (though perhaps arbitrarily small) probability of failure.

Non-singular codes

A code is **non-singular** if each source symbol is mapped to a different non-empty bit string, i.e. the mapping from source symbols to bit strings is injective.

- For example the mapping is **not** non-singular because both "a" and "b" map to the same bit string "0" ; any extension of this mapping will generate a loss (non-lossless) coding. Such singular coding may still be useful when some loss of information is acceptable (for example when such code is used in audio or video compression, where a loss coding becomes equivalent to source quantization).
- However, the mapping is non-singular ; its extension will generate a lossless coding, which will be useful for general data transmission (but this feature is not always required). Note that it is not necessary for the non-singular code to be more compact than the source (and in many applications, a larger code is useful, for example as a way to detect and/or recover from encoding or transmission errors, or in security applications to protect a source from undetectable tampering).

Uniquely Decodable Codes

A code is **uniquely decodable** if its extension is non-singular. Whether a given code is uniquely decodable can be decided with the Sardinas–Patterson algorithm.

- The mapping is uniquely decodable (this can be demonstrated by looking at the *follow-set* after each target bit string in the map, because each bit string is terminated as soon as we see a 0 bit which cannot follow any existing code to create a longer valid code in the map, but unambiguously starts a new code).
- Consider again the code from the previous section. This code, which is based on an example found in,[1] is **not** uniquely decodable, since the string *011101110011* can be interpreted as the sequence of code words *01110–1110 – 011*, but also as the sequence of code words *011 – 1 – 011 – 10011*. Two possible decoding of this encoded string are thus given by *cdb* and *babe*. However, such a code is useful when the set of all possible source symbols is completely known and finite, or when there are restrictions (for example a formal syntax) that determine if source elements of this extension are acceptable. Such restrictions permit the decoding of the original message by checking which of the possible source symbols mapped to the same symbol are valid under those restrictions.

Prefix Codes

A code is a **prefix code** if no target bit string in the mapping is a prefix of the target bit string of a different source symbol in the same mapping. This means that symbols can be decoded instantaneously after their entire code word is received. Other commonly used names for this concept are **prefix-free code**, **instantaneous code**, or **context-free code**.

- The example mapping in the previous paragraph is **not** a prefix code because we don't know after reading the bit string "0" if it encodes an "a" source symbol, or if it is the prefix of the encodings of the "b" or "c" symbols.
- An example of a prefix code is shown below.

Symbol	Code word
A	0
B	10
C	110
D	111

Example of encoding and decoding:

aabacdab → 00100110111010 → |0|0|10|0|110|111|0|10| → aabacdab

A special case of prefix codes are block codes. Here all code words must have the same length. The latter are not very useful in the context of source coding, but often serve as error in the context of channel coding.

Advantages

The advantage of a variable-length code is that unlikely source symbols can be assigned longer code words and likely source symbols can be assigned shorter code words, thus giving a low *expected* code word length. For the above example, if the probabilities of (a, b, c, d) were , the expected number of bits used to represent a source symbol using the code above would be:

.

As the entropy of this source is 1.7500 bits per symbol, this code compresses the source as much as possible so that the source can be recovered with *zero* error.

5.) Different Number Systems

Numeral system (or **system of numeration**) is a writing system for expressing numbers; that is, a mathematical notation for representing numbers of a given set, using digits or other symbols in a consistent manner. It can be seen as the context that allows the symbols "11" to be interpreted as the binary symbol for *three*, the decimal symbol for *eleven*, or a symbol for other numbers in different bases.

The number the numeral represents is called its value.

Ideally, a numeral system will:

- Represent a useful set of numbers (e.g. all integers, or rational numbers)
- Give every number represented a unique representation (or at least a standard representation)
- Reflect the algebraic and arithmetic structure of the numbers.
- For example, the usual decimal representation of whole numbers gives every nonzero whole number a unique representation as a finite sequence of digits, beginning by a non-zero digit. However, when decimal representation is used for the rational or real numbers, such numbers in general have an infinite number of representations, for example 2.31 can also be written as 2.310, 2.3100000, 2.309999999..., etc., all of which have the same meaning except for some scientific and other contexts where greater precision is implied by a larger number of figures shown.
- Numeral systems are sometimes called *number systems*, but that name is ambiguous, as it could refer to different systems of numbers, such as the system of real numbers, the system of complex numbers, the system of *p*-adic numbers, etc. Such systems are, however, not the topic of this article.

5.1) Ancient Number system

The most commonly used system of numerals is the Hindu–Arabic numeral system.[1] Two Indian mathematicians are credited with developing it. Aryabhata of Kusumapura developed the place-value notation in the 5th century and a century later Brahmagupta introduced the symbol for zero. The numeral system and the zero concept, developed by the Hindus in India, slowly spread to other surrounding countries due to their commercial and military activities with India. The Arabs adopted and modified it. Even today, the Arabs call the numerals which they use "Rakam Al-Hind" or the Hindu numeral system. The Arabs translated Hindu texts on numerology and spread them to the western world due to their trade links with them. The Western world modified them and called them the Arabic numerals, as they learned from them. Hence the current western numeral system is the modified version of the Hindu numeral system developed in India. It also exhibits a great similarity to the Sanskrit–Devanagari notation, which is still used in India.

The simplest numeral system is the unary numeral system, in which every natural number is represented by a corresponding number of symbols. If the symbol / is chosen, for example, then the number seven would be represented by ///////. Tally marks represent one such system still in common use. The unary system is only useful for small numbers, although it plays an important role in theoretical computer science. Elias gamma coding, which is commonly used in data compression, expresses arbitrary-sized numbers by using unary to indicate the length of a binary numeral.

Binary Base-2	Decimal Base-10	Hexa-Decimal Base-16	Octal Base-8	BCD Code	Gray Code
0000	0	0	0	0	0000
0001	1	1	1	1	0001
0010	2	2	2	2	0011
0011	3	3	3	3	0010
0100	4	4	4	4	0110
0101	5	5	5	5	0111
0110	6	6	6	6	0101
0111	7	7	7	7	0100
1000	8	8	10	8	1100
1001	9	9	11	9	1101
1010	10	A	12	---	1111
1011	11	B	13	---	1110
1100	12	C	14	---	1010
1101	13	D	15	---	1011
1110	14	E	16	---	1001
1111	15	F	17	---	1000

5.2) Unary number system

The **unary numeral system** is the bijective base-1 numeral system. It is the simplest numeral system to represent natural numbers:[1] in order to represent a number *N*, an arbitrarily chosen symbol representing 1 is repeated *N* times.[2] For examples, the numbers 1, 2, 3, 4, 5, ... would be represented in this system as[3]

1, 11, 111, 1111, 11111, ...

These numbers should be distinguished from repunits, which are also written as sequences of ones but have their usual decimal numerical interpretation.

This system is used in tallying. For example, using the tally mark **|**, the number 3 is represented as **|||**. In East Asian cultures, the number three is represented as "三", a character that is drawn with three strokes.[4]

5.3) Decimal (Base 10) Number System

Decimal number system has ten symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, called *digit*s. It uses *positional notation*. That is, the least-significant digit (right-most digit) is of the order of 10^0 (units or ones), the second right-most digit is of the order of 10^1 (tens), the third right-most digit is of the order of 10^2 (hundreds), and so on. For example,

735 = 7×10^2 + 3×10^1 + 5×10^0

5.4) Binary (Base 2) Number System

Binary number system has two symbols: 0 and 1, called *bits*. It is also a *positional notation*, for example,

10110B = 1×2^4 + 0×2^3 + 1×2^2 + 1×2^1 + 0×2^0

We shall denote a binary number with a suffix B. Some programming languages denote binary numbers with prefix 0b (e.g., 0b1001000), or prefix b with the bits quoted (e.g., b'10001111').

A binary digit is called a *bit*. Eight bits is called a *byte* (why 8-bit unit? Probably because $8=2^3$).

5.5) Binary-coded decimal

In computing and electronic systems, **binary-coded decimal** (**BCD**) is a class of binary encodings of decimal numbers where each decimal digit is represented by a fixed number of bits, usually four or eight. Special bit patterns are sometimes used for a sign or for other indications (e.g., error or overflow).

In byte-oriented systems (i.e. most modern computers), the term *unpacked* BCD[1] usually implies a full byte for each digit (often including a sign), whereas *packed* BCD typically encodes two decimal digits within a single byte by taking advantage of the fact that four bits are enough to represent the range 0 to 9. The precise 4-bit encoding may vary however, for technical reasons, see Excess-3 for instance. The ten states representing a BCD decimal digit are sometimes called *tetrads* (for the nibble typically needed to hold them also known as tetrad) with those don't care-states unused named *pseudo-tetrad(e)s* (de) or *pseudo-decimal digit*[6]).

BCD was used in many early decimal computers, and is implemented in the instruction set of machines such as the IBM System/360 series and its descendants and Digital's VAX. Although BCD *per se* is not as widely used as in the past and is no longer implemented in computers' instruction sets, decimal fixed-point and floating-point formats are still important and continue to be used in financial, commercial, and industrial computing, where subtle conversion and fractional rounding errors that are inherent in floating point binary representations cannot be tolerated.

5.6) Hexadecimal (Base 16) Number System

Hexadecimal number system uses 16 symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F, called *hex digits*. It is a *positional notation*, for example,

A3EH = 10×16^2 + 3×16^1 + 14×16^0

We shall denote a hexadecimal number (in short, hex) with a suffix H. Some programming languages denote hex numbers with prefix 0x (e.g., 0x1A3C5F), or prefix x with hex digit quoted (e.g., x'C3A4D98B').

Each hexadecimal digit is also called a *hex digit*. Most programming languages accept lowercase 'a' to 'f' as well as uppercase 'A' to 'F'.

Computers uses binary system in their internal operations, as they are built from binary digital electronic components. However, writing or reading a long sequence of binary bits is cumbersome and error-prone. Hexadecimal system is used as a *compact* form or *shorthand* for binary bits. Each hex digit is equivalent to 4 binary bits, i.e., shorthand for 4 bits, as follows:

0H (0000B) (0D)	1H (0001B) (1D)	2H (0010B) (2D)	3H (0011B) (3D)
4H (0100B) (4D)	5H (0101B) (5D)	6H (0110B) (6D)	7H (0111B) (7D)
8H (1000B) (8D)	9H (1001B) (9D)	AH (1010B) (10D)	BH (1011B) (11D)
CH (1100B) (12D)	DH (1101B) (13D)	EH (1110B) (14D)	FH (1111B) (15D)

6.) FORMAL DEFINITION

A n bit Binary Number is $a_n a_{n-1} a_{n-2} \ldots a_1 a_0$ where each $a_i \varepsilon \{0, 1\}$ and $1 \leq I \leq n$.

Examples: $(101110)_2$, $(101.10110)_2$

A n bit Decimal Number is $a_n a_{n-1} a_{n-2} \ldots . a_1 a_0$ where each $a_i \varepsilon \{ 0 ,1 , 2 , . 9\}$ and $1 \leq I \leq n$.

Examples: $(1526)_{10}$, $(12.45)_{10}$ etc.

A n bit Octal Number is $a_n a_{n-1} a_{n-2} \ldots . a_1 a_0$ where each $a_i \varepsilon \{ 0 ,1 , 2 , \ldots 7\}$ and $1 \leq I \leq n$.

Examples: $(257)_8$, $(267.012)_8$

A n bit Hexadecimal Number is $a_n a_{n-1} a_{n-2} \ldots a_1 a_0$ where each $a_i \varepsilon \{0 ,1, 2, \ldots 9 ,A , B , C , D , E , F \}$ and $1 \leq I \leq n$.

Examples: $(F23A)_{16}$, $(F23.25A)_{16}$

DECIMAL	BINARY	OCTAL	HEXA DECIMAL
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	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
16	10000	20	10

16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
22	10110	26	16
23	10111	27	17
24	11000	30	18
25	11001	31	19
26	11010	32	1A
27	11011	33	1B
28	11100	34	1C
29	11101	35	1D
30	11110	36	1E
31	11111	37	1F
32	100000	40	20

7.) Conversion from Different to Different Number Systems

General rule

Any numbering system can be summarised by the following relationship:

$$\underline{N = b_i q^i}$$

where:

N is real positive number
b is the digit
q is the base value
and integer I can be positive, negative or zero

$$N = b_n q^n \ldots b_3 q^3 + b_2 q^2 + b_1 q^1 + b_0 q^0 + b_{-1} q^{-1} + b_{-2} q^{-2} \ldots \text{etc.}$$

7.1) BINARY TO DECIMAL

We saw above that in the decimal number system, the weight of each digit to the left increases by a factor of 10. In the binary number system, the weight of each digit increases by a factor of 2 as shown. Then the first digit has a weight of 1 (2^0), the second digit has a weight of 2 (2^1), the third a weight of 4 (2^2), the fourth a weight of 8 (2^3) and so on.

So for example, converting a **Binary to Decimal** number would be:

Decimal Digit Value	256	128	64	32	16	8	4	2	1
Binary Digit Value	1	0	1	1	0	0	1	0	1

By adding together ALL the decimal number values from right to left at the positions that are represented by a "1" gives us: (256) + (64) + (32) + (4) + (1) = 357_{10} or three hundred and fifty-seven as a decimal number.

Then, we can convert binary to decimal by finding the decimal equivalent of the binary array of digits

101100101_2 and expanding the binary digits into a series with a base of 2 giving an equivalent of 357_{10} in decimal or denary.

BINARY	1	1	0	1
Face Value	8	4	2	1
Decimal Equivalent	8	4	0	1

TOTAL =	13

7.2) OCTAL TO DECIMAL

1. Start the decimal result at 0.
2. Remove the most significant octal digit (leftmost) and add it to the result.
3. If all octal digits have been removed, you're done. Stop.
4. Otherwise, multiply the result by 8.
5. Go to step 2.

OCTAL	6	3	4	7	4
BINARY	110	011	100	111	100

7.3) HEXADECIMAL TO DECIMAL

Hex is a base 16 number and decimal is a base 10 number. We need to know the decimal equivalent of every hex number digit. See below of the page to check the hex to decimal chart.
Here are the steps to convert hex to decimal:

Get the decimal equivalent of hex from table.
Multiply every digit with 16 power of digit location.
(zero based, 7DE: E location is 0, D location is 1 and the 7 location is 2)
Sum all the multipliers.

Here is an example:

```
7DE is a hex number
7DE = (7 * 16^2) + (13 * 16^1) + (14 * 16^0)
7DE = (7 * 256) + (13 * 16) + (14 * 1)
7DE = 1792 + 208 + 14
7DE = 2014 (in decimal number)
```

HEXA-DECIMAL	D	C	9	8	4
BINARY	1101	1100	1001	1000	0100

7.4)Decimal To Binary

Repeated Division-by-2 Method

We have seen above how to convert binary to decimal numbers, but how do we convert a decimal number into a binary number. An easy method of converting decimal to binary number equivalents is to write down the decimal number and to continually divide-by-2 (two) to give a result and a remainder of either a "1" or a "0" until the final result equals zero.

So for example. Convert the decimal number 294_{10} into its binary number equivalent.

Number 294

divide by 2

result 147 remainder **0** (LSB)

divide by 2

result 73 remainder **1**

divide by 2

result 36 remainder **1**

divide by 2

result 18 remainder **0**

divide by 2

result 9 remainder **0**

Dividing each decimal number by "2" as shown will give a result plus a remainder.

If the decimal number being divided is even, then the result will be whole and the remainder will be equal to "0". If the decimal number is odd, then the result will not divide completely and the remainder will be a "1".

The binary result is obtained by placing all the remainders in order with the least significant bit (LSB) being at the top and the most

divide by 2

result 4 remainder **1**

divide by 2

result 2 remainder **0**

divide by 2

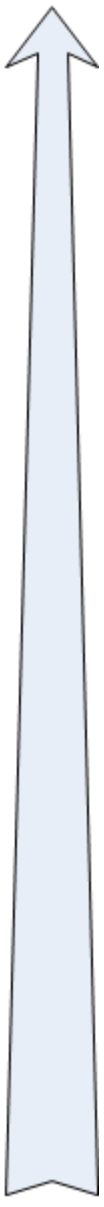
result 1 remainder **0**

divide by 2

result 0 remainder **1** (MSB)

significant bit (MSB) being at the bottom.

2	1100	
2	550	0
2	275	0
2	137	1
2	68	1
2	34	0
2	17	0
2	8	1
2	4	0
2	2	0
2	1	0
2	0	1



(Converting 1100 from decimal to binary using repeated division by 2. Its binary representation is 10001001100)

This divide-by-2 decimal to binary conversion technique gives the decimal number 294_{10} an equivalent of 100100110_2 in binary, reading from right to left. This divide-by-2 method will also work for conversion to other number bases.

Then we can see that the main characteristics of a **Binary Numbering System** is that each "binary digit" or "bit" has a value of either "1" or "0" with each bit having a weight or value double that of its previous bit starting from the lowest or least significant bit (LSB) and this is called the "sum-of-weights" method.

So we can convert a decimal number into a binary number either by using the sum-of-weights method or by using the repeated division-by-2 method, and convert binary to decimal by finding its sum-of-weights.

Correctness of Algorithm

Let's say we have a number, say with binary expansion 1101010101011.

First, let's find the last digit. If the number is even then the last digit is zero, and if the number is odd then the last digit is one, so we can determine the last digit by dividing by 2 and looking at the remainder:

1101010101011 = 10 * 110101010101 + 1

To find the next-to-last digit, just find the last digit of 110101010101 by the same method, and so forth.

7.5) Hexadecimal to Binary

Replace each hex digit by the 4 equivalent bits, for examples,

A3C5H = 1010 0011 1100 0101B

102AH = 0001 0000 0010 1010B

HEXA-DECIMAL	D	C	9	8	4
BINARY	1101	1100	1001	1000	0100

7.6) Octal to Binary

Steps

Step 1 - Convert each octal digit to a 3-digit binary number (the octal digits may be treated as decimal for this conversion).

Step 2 - Combine all the resulting binary groups (of 3 digits each) into a single binary number.

OCTAL	6	3	4	7	4
BINARY	110	011	100	111	100

7.7) Binary to Hexadecimal

Starting from the right-most bit (least-significant bit), replace each group of 4 bits by the equivalent hex digit (pad the left-most bits with zero if necessary), for examples,

1001001010B = 0010 0100 1010B = 24AH

10001011001011B = 0010 0010 1100 1011B = 22CBH

It is important to note that hexadecimal number provides a *compact form* or *shorthand* for representing binary bits.

BINARY	1101	1100	1001	1000	0100
HEXA-DECIMAL	D	C	9	8	4

7.8) Octal to Hexadecimal

When converting from octal to hexadecimal, it is often easier to first convert the octal number into binary and then from binary into hexadecimal. For example, to convert 345 octal into hex:

(from the previous example)

Octal = 3 4 5

Binary = 011 100 101 = 011100101 binary

Drop any leading zeros or pad with leading zeros to get groups of four binary digits (bits):
Binary 011100101 = 1110 0101

Then, look up the groups in a table to convert to hexadecimal digits.

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

Binary = 1110 0101
Hexadecimal = E 5 = E5 hex

Therefore, through a two-step conversion process, octal 345 equals binary 011100101 equals hexadecimal E5

OCTAL	3	3	4	4	6	0	4
BINARY	011	011	100	100	110	000	100

Converting the binary number into groups of four

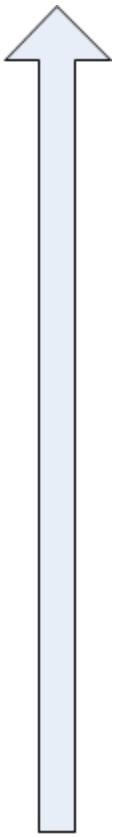
BINARY	1101	1100	1001	1000	0100
HEXA-DECIMAL	D	C	9	8	4

7.9) Decimal to Hexadecimal

Steps:

I. Divide the decimal number by 16. Treat the division as an integer division.
II. Write down the remainder (in hexadecimal).
III. Divide the result again by 16. Treat the division as an integer division.
IV. Repeat step 2 and 3 until result is 0.
V. The hex value is the digit sequence of the remainders from the last to first.

16	941236	
16	58827	4
16	3676	11
16	229	12
16	14	5
16	0	14



7.10) Binary to octal

Steps

- **Step 1 -** Divide the binary digits into groups of three (starting from the right).
- **Step 2 -** Convert each group of three binary digits to one octal

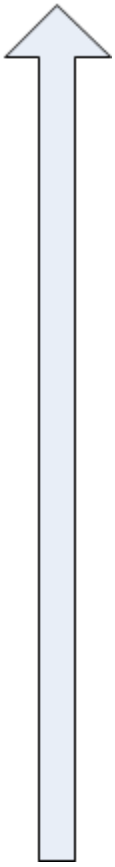
BINARY	011	011	100	100	110	000	100
OCTAL	3	3	4	4	6	0	4

7.11) Decimal to octal

Any decimal number can be converted into octal number system by dividing it by the largest possible power of 8. The remainder then again divided by the largest possible power of 8, this method is repeated until the power of 8 is 1 and the remainder obtained is less than 8. The result is obtained by arranging the obtained quotients as they are acquired. The method will be clear by a

simple example. Let us convert 13010 into octal number First of all we have to divide it by the largest possible power of 8 i.e. 82 130 / 82 = 2 130 - 82*2 = 2 2 / 81 = 0 2 - 0 = 2 2/80 = 2 So the octal equivalent of 13010 is 2028

8	56789	
8	7098	5
8	887	2
8	110	7
8	13	6
8	1	5
8	0	1



7.12) Hexadecimal to octal

This below example is the basic principle used in this Hexadecimal to Octal conversion. The easiest way to convert Hex number to octal number is converting hex number to its equivalent binary number and convert it to its equivalent octal number for example the equivalent Octal number for 25BH can be derived as Hex to Binary Conversion 2 = 0010; 5 = 0101; B = 1011 The Binary Number = 0010 0101 1011 Now the hex decimal equivalent Binary numbers can be converted into equivalent octal numbers. To do so, split the binary number into segments having 3 bits each to find out the equivalent octal for the binary number 001 = 1; 001 = 1; 011 = 3; 011 = 3 The Octal Number = 11338

HEXA-DECIMAL	D	C	9	8	4
BINARY	1101	1100	1001	1000	0100

Converting the binary number into groups of three

BINARY	011	011	100	100	110	000	100
OCTAL	3	3	4	4	6	0	4

7.13) Base *r* to Decimal (Base 10)

Given a *n*-digit base *r* number: dn-1 dn-2 dn-3 ... d3 d2 d1 d0 (base r), the decimal equivalent is given by:

dn-1 × r^(n-1) + dn-2 × r^(n-2) + ... + d1 × r^1 + d0 × r^0

For examples,

A1C2H = 10×16^3 + 1×16^2 + 12×16^1 + 2 = 41410 (base 10)

10110B = 1×2^4 + 1×2^2 + 1×2^1 = 22 (base 10)

Given a *n*-digit base *r* number: dn-1 dn-2 dn-3 ... d3 d2 d1 d0 (base r), the decimal equivalent is given by:

dn-1 × r^(n-1) + dn-2 × r^(n-2) + ... + d1 × r^1 + d0 × r^0

For examples,

A1C2H = 10×16^3 + 1×16^2 + 12×16^1 + 2 = 41410 (base 10)

10110B = 1×2^4 + 1×2^2 + 1×2^1 = 22 (base 10)

FOR r=3 BASE-R	2	1	1	0	2
FACE VALUE	81	27	9	3	1
DECIMAL	162	27	9	0	2
				TOTAL =	200

7.14) Decimal (Base 10) to Base *r*

Use repeated division/remainder. For example,

To convert 261D to hexadecimal:

261/16 => quotient=16 remainder=5

16/16 => quotient=1 remainder=0

1/16 => quotient=0 remainder=1 (quotient=0 stop)

Hence, 261D = 105H

The above procedure is actually applicable to conversion between any 2 base systems. For example,

To convert 1023(base 4) to base 3:

1023(base 4)/3 => quotient=25D remainder=0

25D/3 => quotient=8D remainder=1

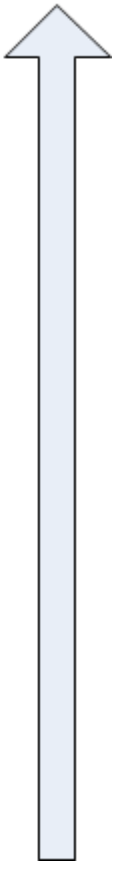
8D/3 => quotient=2D remainder=2

2D/3 => quotient=0 remainder=2 (quotient=0 stop)

Hence, 1023(base 4) = 2210(base 3).

DECIMAL TO BASE 3

3	700	
3	233	1
3	77	2
3	25	2
3	8	1
3	2	2
3	0	2



8.) General Conversion between 2 Base Systems with Fractional Part

1. Separate the integral and the fractional parts.
2. For the integral part, divide by the target radix repeatable, and collect the remainder in reverse order.
3. For the fractional part, multiply the fractional part by the target radix repeatable, and collect the integral part in the same order.

Example 1:

Convert 18.6875D to binary

Integral Part = 18D

18/2 => quotient=9 remainder=0

9/2 => quotient=4 remainder=1

4/2 => quotient=2 remainder=0

2/2 => quotient=1 remainder=0

1/2 => quotient=0 remainder=1 (quotient=0 stop)

Hence, 18D = 10010B

.6875	x2	1.375	1
.375	x2	0.75	0
.75	x2	1.50	1
.50	x2	1	1

Fractional Part = .6875D

.6875*2=1.375 => whole number is 1

.375*2=0.75 => whole number is 0

.75*2=1.5 => whole number is 1

.5*2=1.0 => whole number is 1

Hence .6875D = .1011B

Therefore, 18.6875D = 10010.1011B

Example 2:

Convert 18.6875D to hexadecimal

Integral Part = 18D

18/16 => quotient=1 remainder=2

1/16 => quotient=0 remainder=1 (quotient=0 stop)

Hence, 18D = 12H

Fractional Part = .6875D

.6875*16=11.0 => whole number is 11D (BH)

Hence .6875D = .BH

Therefore, 18.6875D = 12.BH

You already know how to convert whole numbers from decimal to binary. So think about what a decimal really means. You have a sum of different amounts of powers of 10.

Example 3, 428.4 is actually:

(102

* 4) + (101 * 2) + (101 * 8) + (10−1

* 4)

So do the same thing. Except instead of having the powers of 10, you use powers of 2. In other words, instead of 10ths 100ths and 1000ths columns, you have halves, fourths, and eighths.

So let's convert 12.510

to binary

Write it as a sum of powers of ten:

(101

* 1) + (100 * 2) + (10−1

* 5)

Now write it as a sum of powers of two:

(23

* 1) + (22 * 1) + (21 * 0) + (20 * 0) + (2−1

* 1)

12.510

= 1100.12

The reason a lot of students have a problem with this is because many "simple" numbers may look very complex when converted.

For example: Convert 0.3 to a sum of powers of 2. It can be done, but the binary expansion is going to have an infinitely repeating decimal.

So the "cheat" way to do it, (for rational numbers at least) is to convert it into a base 10 fraction. And then convert both the numerator and denominator into binary.

0.3 = 3101010

= 11210102

9.) LSB, MSB, Sign Bit

In computing, the least significant bit (LSB) is the bit position in a binary integer giving the unit's value, that is, determining whether the number is even or odd. The LSB is sometimes referred to as the right-most bit, due to the convention in positional notation of writing less significant digits farther to the right. It is analogous to the least significant digit of a decimal integer, which is the digit in the ones (right-most) position. [1]

In computing, the most significant bit (MSB, also called the high-order bit) is the bit position in a binary number having the greatest value. The MSB is sometimes referred to as the left-most bit due to the convention in positional notation of writing more significant digits further to the left.

The MSB can also correspond to the sign bit of a signed binary number in one's or two's complement notation, "1" meaning negative and "0" meaning positive.

In computer science, the sign bit is a bit in a signed number representation that indicates the sign of a number. Although only signed numeric data types have a sign bit, it is invariably located in the most significant bit position, so the term may be used interchangeably with "most significant bit" in some contexts.

Almost always, if the sign bit is 0, the number is non-negative (positive or zero). If the sign bit is 1 then the number is negative, although formats other than two's complement integers allow a signed zero: distinct "positive zero" and "negative zero" representations, the latter of which does not correspond to the mathematical concept of a negative number.

END