Ascii :

Es un código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno. Fue creado en 1963 por el Comité Estadounidense de Estándares (ASA, conocido desde 1969 como el Instituto Estadounidense de Estándares Nacionales, o ANSI) como una evolución de los conjuntos de códigos utilizados entonces en telegrafía.

El código ASCII utiliza 7 bits para representar los caracteres, aunque inicialmente empleaba un bit adicional (bit de paridad) que se usaba para detectar errores en la transmisión.

El código ASCII reserva los primeros 32 códigos (numerados del 0 al 31 en decimal) para caracteres de control: códigos no pensados originalmente para representar información imprimible, sino para controlar dispositivos (como impresoras) que usaban ASCII. Por ejemplo, el carácter 10 representa la función "nueva línea" (line feed), que hace que una impresora avance el papel.

Sistemas numericos :

Octal:

Definición General

El sistema octal es un sistema de numeración base 8 que utiliza ocho dígitos (0-7). Es ampliamente empleado en computación para representar datos binarios de manera más compacta, especialmente en ciertos sistemas de archivos y permisos de UNIX/Linux.

Características Principales

1. **Conjunto de Dígitos**:
    - Solo incluye los números del **0** al **7**.
    - Cada dígito octal equivale a 3 bits en binario.
2. **Conversión a Binario**:
    - Octal agrupa bits en múltiplos de 3, lo que facilita la conversión directa a binario:
        - **Octal → Binario**: Cada dígito octal se convierte en un grupo de 3 bits.
        - **Binario → Octal**: Los bits se agrupan en bloques de 3 (de derecha a izquierda) y cada grupo se convierte en un dígito octal.

Ejemplo:

- Binario: `110110` → Agrupando en bloques de 3: `110 110` → Octal: **66**.
- Octal: `7` → Binario: **111**.

3. **Representación de Datos en Octal**:
    - En programación, los números octales suelen ser precedidos por un prefijo para diferenciarlos de otras bases:
        - Python: `0o123` (123 en octal).
        - C: `0123` (123 en octal).

Codificación y Decodificación

1. **Codificación en Octal**:

- Se convierte el valor binario o ASCII de cada carácter a su representación en octal.
- Ejemplo (ASCII → Octal):
  - La letra "A" tiene un valor ASCII de 65.
  - En binario: `01000001`.
  - En octal: `101`.

2. **Decodificación desde Octal**:
   - El proceso inverso: cada dígito octal se convierte a binario y se reconstruyen los datos originales.
   - Ejemplo:
     - Octal: `101`.
     - Binario: `01000001`.
     - ASCII: "A".

Ventajas

1. Compacta la representación binaria en bloques más cortos, facilitando la lectura y escritura manual.
2. Es útil en sistemas de archivos UNIX/Linux, donde se utiliza para representar permisos (lectura, escritura, ejecución) de manera eficiente.

Limitaciones

1. Menos eficiente en términos de compresión comparado con hexadecimal (base 16).
2. Poco usado fuera de aplicaciones específicas como permisos de sistemas de archivos.

Usos Comunes

1. **Permisos en UNIX/Linux**:
   - Cada permiso se representa como un dígito octal:
     - `rwx` (lectura, escritura, ejecución) → Binario: `111` → Octal: `7`.
     - Un archivo con permisos `rwxr--r--` sería `744` en octal.
2. **Codificación de Caracteres**:
   - Representación de caracteres especiales en cadenas o scripts, como `\033` para escape.
3. **Sistemas Integrados**:
   - Algunos sistemas embebidos o lenguajes de programación usan octal para representar direcciones de memoria o datos.

Hexadecimal:

Definición General

El sistema hexadecimal (base 16) es un sistema de numeración que utiliza 16 símbolos: los números del 0 al 9 y las letras de la A a la F, donde:

- **A = 10**, **B = 11**, **C = 12**, **D = 13**, **E = 14**, y **F = 15**.

Se usa comúnmente en computación y programación para representar datos binarios de manera más compacta y eficiente.

Características Principales

1. **Conjunto de Dígitos**:
    - Incluye los números del **0** al **9** y las letras **A-F**.
    - Cada dígito hexadecimal representa 4 bits (nibble) en binario.
2. **Conversión a Binario**:
    - Hexadecimal → Binario: Cada dígito hexadecimal se convierte en un grupo de 4 bits.
    - Binario → Hexadecimal: Los bits se agrupan en bloques de 4 (de derecha a izquierda) y se convierten en un dígito hexadecimal.

Ejemplo:

- Binario: `11011011` → Agrupando en bloques de 4: `1101 1011` → Hexadecimal: **DB**.
- Hexadecimal: `F` → Binario: **1111**.

3. **Representación de Datos en Hexadecimal**:
    - En lenguajes de programación, los números hexadecimales suelen tener prefijos:
        - Python: `0x1A3` (1A3 en hexadecimal).
        - C: `0x1A3`.
        - Ensamblador: `1A3h`.

Codificación y Decodificación

1. **Codificación a Hexadecimal**:
    - Se convierte cada byte (8 bits) o valor ASCII a su representación hexadecimal.
    - Ejemplo (ASCII → Hexadecimal):
        - La letra "H" tiene un valor ASCII de 72.
        - En binario: `01001000`.
        - En hexadecimal: **48**.
2. **Decodificación desde Hexadecimal**:
    - El proceso inverso: cada dígito hexadecimal se convierte a binario, y luego se reconstruyen los datos originales.
    - Ejemplo:
        - Hexadecimal: **48**.
        - Binario: `01001000`.
        - ASCII: "H".

Ventajas

1. **Compacidad**:
    - Representa grandes cantidades de datos binarios con menos dígitos.

- Ejemplo: `1111111111111111` (16 bits en binario) → `FFFF` en hexadecimal.

2. **Legibilidad**:
    - Más fácil de leer y escribir en comparación con binario.
    - Comúnmente utilizado en depuración y análisis de datos binarios.

Limitaciones

1. No es tan intuitivo como los sistemas decimales para cálculos mentales.
2. Requiere aprendizaje para interpretar valores A-F.

Usos Comunes

1. **Direcciones de Memoria y Datos Binarios**:
    - Las direcciones en memoria, colores en HTML (por ejemplo, `#FF5733`), y datos en bajo nivel suelen expresarse en hexadecimal.
2. **Representación de Códigos ASCII y Unicode**:
    - Ejemplo: El carácter `A` es **41** en hexadecimal.
3. **Cifrado y Codificación**:
    - Hexadecimal se utiliza para representar datos cifrados (hashes, claves) de manera compacta.

Base 64:

Definición General

Base64 es un método de codificación que permite representar datos binarios en un formato de texto legible utilizando únicamente caracteres ASCII. Este esquema es particularmente útil para transmitir datos binarios (como imágenes o archivos) a través de sistemas diseñados para manejar texto, como correos electrónicos o JSON.

Características Principales

1. **Conjunto de Caracteres**: Base64 utiliza 64 caracteres:
    - Letras mayúsculas: `A-Z` (valores 0–25).
    - Letras minúsculas: `a-z` (valores 26–51).
    - Dígitos: `0-9` (valores 52–61).
    - Dos símbolos: + (valor 62) y `/` (valor 63).
    - Un carácter de relleno (=) se utiliza para ajustar la longitud.
2. **Bloques de Datos**:
    - Base64 procesa los datos en bloques de **3 bytes** (24 bits), dividiéndolos en 4 bloques de 6 bits cada uno.
    - Si la cantidad de datos no es múltiplo de 3, se agregan bytes de relleno (`\0`), y el carácter = se utiliza en la salida para indicar el ajuste.
3. **Codificación**:

- Los datos se convierten en su representación binaria.
- Se agrupan en bloques de 6 bits, que se mapean al conjunto de caracteres de Base64.
- Por ejemplo, el texto "Man" se codifica en Base64 como **"TWFu"**:
  - "Man" en ASCII: `01001101 01100001 01101110` (24 bits).
  - Dividido en 6 bits: `010011 010110 000101 101110`.
  - Mapeado a Base64: `T W F u`.

4. **Decodificación**:
   - Se realiza el proceso inverso:
     - Cada carácter de Base64 se convierte en su valor binario de 6 bits.
     - Los bloques de 6 bits se agrupan para formar bloques de 8 bits.
     - El resultado final es la reconstrucción de los datos originales.

Ventajas

- Es compatible con sistemas que manejan texto ASCII exclusivamente.
- Preserva la integridad de los datos binarios en entornos no binarios (como emails o URIs).

Limitaciones

- Incrementa el tamaño de los datos en un 33%. Por ejemplo, 3 bytes originales se codifican como 4 caracteres en Base64.
- No es un mecanismo de cifrado, ya que su propósito no es proteger los datos, sino solo transformarlos.

Usos Comunes

- **Correos electrónicos**: En el estándar MIME, Base64 se utiliza para transmitir archivos adjuntos.
- **Autenticación HTTP**: En encabezados de autorización.
- **JSON y APIs**: Para serializar datos binarios en texto.
- **Codificación de imágenes y multimedia**: Integración de datos directamente en documentos HTML o CSS (como imágenes en línea con `data:` URLs).

Decimal:

Definición General
El sistema decimal (base 10) es el sistema numérico más comúnmente utilizado por los seres humanos. Se basa en diez dígitos: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**, con un valor posicional determinado por potencias de 10.
Características Principales

1. **Conjunto de Dígitos**:
   - El sistema decimal utiliza diez símbolos únicos, desde el 0 al 9.
2. **Sistema Posicional**:

- El valor de cada dígito depende de su posición en el número, según potencias de 10.
- Ejemplo: En el número **753**, el valor se calcula como: $(7\times10^2)+(5\times10^1)+(3\times10^0)=700+50+3 =753$.

3. **Relación con Otros Sistemas**:
    - Es la base de conversión para sistemas binarios, octales, y hexadecimales.
    - Conversión de decimal a otra base: Se divide el número decimal por la base de destino, tomando los residuos.
    - Conversión de otra base a decimal: Se suman los dígitos multiplicados por potencias de su base.

Codificación y Decodificación

1. **Codificación Decimal**:
    - Los datos en otros sistemas (como binario o hexadecimal) pueden representarse en decimal.
    - Ejemplo: En binario, 110121101_211012 = $(1\times23)+(1\times22)+(0\times21)+(1\times20)=13$ en decimal.
2. **Decodificación Decimal**:
    - Para decodificar valores decimales, se convierten a sus sistemas de origen (por ejemplo, ASCII para caracteres).
    - Ejemplo: El número decimal 65 se decodifica como el carácter `A` en el estándar ASCII.

Ventajas

1. **Intuición y Usabilidad**:
    - Es el sistema más intuitivo para los humanos debido a su uso en la vida cotidiana.
2. **Flexibilidad para Cálculos**:
    - Es ideal para operaciones aritméticas y representación de cantidades en contextos generales.

Limitaciones

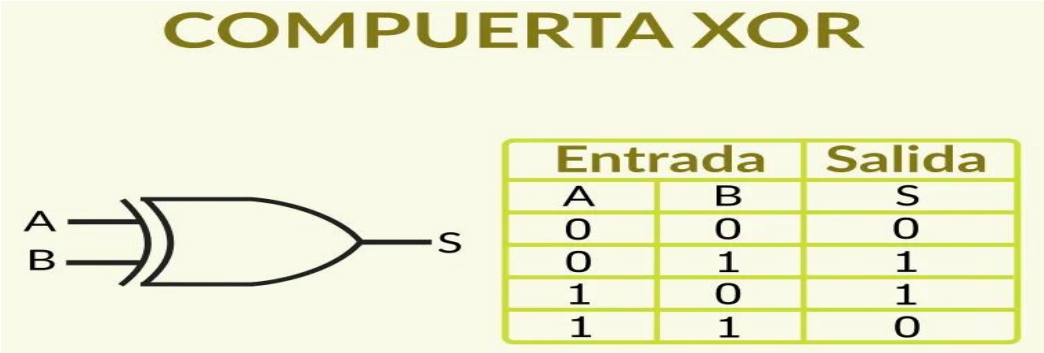
1. **Ineficiencia en Computación**:
    - No es eficiente para representar valores a nivel de hardware, donde se prefieren sistemas binarios.
2. **Conversión con Pérdidas**:
    - En algunos casos, como números fraccionarios, puede no representar exactamente valores en otras bases.

Usos Comunes

1. **Representación de Cantidades**:
    - Es la base de representación numérica para mediciones, cálculos matemáticos y contabilidad.

2. **Interfaz Humana en Computación**:
    - Aunque las computadoras operan en binario, los valores se traducen a decimal para facilitar la interacción humana.
3. **Conversiones y Codificaciones**:
    - Los sistemas computacionales traducen entre decimal y otras bases (como hexadecimal o ASCII).

XOR



Definición General
XOR (abreviatura de Exclusive OR) es una operación lógica binaria que devuelve un resultado verdadero (1) si y solo si los bits de entrada son diferentes. Matemáticamente, se define como:

$$A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$$

En criptografía, XOR se utiliza ampliamente debido a sus propiedades algebraicas, que lo hacen ideal para mezclar datos en procesos de codificación y decodificación.

Propiedades Fundamentales del XOR:

Conmutativa:

$$A \oplus B = B \oplus A$$

Asociativa:

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C$$

Elemento Neutro:

$$A \oplus 0 = A$$

Involutiva:

$$A \oplus A = 0$$

Esto significa que aplicar XOR dos veces con el mismo valor devuelve el valor original.

Codificación con XOR
En criptografía, el XOR se utiliza para cifrar datos al combinar el texto plano con una clave. El proceso se puede describir como:

$$\text{Texto Cifrado} = \text{Texto Plano} \oplus \text{Clave}$$

Ventajas:
Es extremadamente rápido y eficiente para implementar tanto en hardware como en software.
Puede proporcionar seguridad básica en sistemas ligeros o como componente de algoritmos más complejos.

Decodificación con XOR
Dado que $A \oplus B \oplus B = A$ el mismo XOR se utiliza para descifrar el texto:

$$\text{Texto Plano} = \text{Texto Cifrado} \oplus \text{Clave}$$

Ventajas y Desventajas de XOR en Criptografía

Ventajas:

- Simplicidad: Fácil de implementar y extremadamente eficiente.
- Reversibilidad: Perfecto para codificación y decodificación.

Desventajas:

- Seguridad débil si se reutiliza la clave (ataque conocido como *ataque de texto plano conocido*).
- Dependencia de claves verdaderamente aleatorias para proteger la confidencialidad.

Ejemplo Práctico: Ataques con XOR
En criptografía, si se usa la misma clave para cifrar dos textos diferentes, se puede derivar información combinando los textos cifrados:

$$C1 \oplus C2 = (\text{Texto Plano 1} \oplus \text{Clave}) \oplus (\text{Texto Plano 2} \oplus \text{Clave}) = \text{Texto Plano 1} \oplus \text{Texto Plano 2}$$

Este resultado puede ser explotado por un atacante para descubrir patrones.

Conclusión
XOR es un componente crítico en muchos sistemas criptográficos debido a su simplicidad y propiedades únicas. Sin embargo, es importante combinarlo con otros mecanismos criptográficos y manejar cuidadosamente las claves para evitar vulnerabilidades.