
**FACULTAD DE CIENCIAS DE LA COMPUTACIÓN Y DISEÑO DIGITAL**

**INTEGRANTES**

GAMARRA ARAUJO EDHU XAVIER

MENDOZA PARRAGA ANDY JOHEL

NARANJO FLORES ANDERSON JEAMPIERE

**CURSO**

2DO SOFTWARE “B”

**GRUPO**

B

**MATERIA**

ARQUITECTURA DE COMPUTADORAS

**TEMA**

# SISTEMAS NUMÉRICOS Y REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

# ÍNDICE

# SISTEMAS NUMÉRICOS Y REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

## OBJETIVO

Desarrollar habilidades para realizar operaciones en sistemas numéricos binario, octal y hexadecimal, así como entender el flujo y representación de datos dentro de una computadora.

## FUNDAMENTO TEÓRICO

### 1. Operaciones Fundamentales en Binario

El sistema de numeración binario, que están conformado solo por 0 y 1, es el pilar fundamental de los sistemas informáticos. Debido a su simplicidad y estar relacionado directamente con el estado de los dispositivos electrónicos (encendido o apagado), el sistema binario permite representar datos, realizar operaciones lógicas y controlar procesos digitales [1].

El sistema binario emplea el mismo principio de valor posicional que el sistema decimal, con la diferencia de que en lugar de usar la base 10, el sistema binario utiliza la base 2 conformada por 0 y 1. Para determinar el valor de cada posición de un número binario, se multiplica la potencia de la base que en este caso es 2 [2]. A continuación, en las Tablas 1 y 2 se muestran los valores decimales vinculados a cada posición de un número binario tanto en su parte entera como fraccionaria:

**Tabla 1.** Potencias Positivas

| **Potencias positivas** | |
|---|---|
| $\mathbf{2^0}$ | 1 |
| $\mathbf{2^1}$ | 2 |
| $\mathbf{2^2}$ | 4 |
| $\mathbf{2^3}$ | 8 |
| $\mathbf{2^4}$ | 16 |
| $\mathbf{2^5}$ | 32 |
| $\mathbf{2^6}$ | 64 |
| $\mathbf{2^7}$ | 128 |
| $\mathbf{2^8}$ | 256 |

**Tabla 2.** Potencias negativas

| **Potencias negativas** | |
|---|---|
| $\mathbf{2^{-1}}$ | 0.5 |
| $\mathbf{2^{-2}}$ | 0.25 |
| $\mathbf{2^{-3}}$ | 0.125 |
| $\mathbf{2^{-4}}$ | 0.0625 |
| $\mathbf{2^{-5}}$ | 0.03125 |
| $\mathbf{2^{-6}}$ | 0.015625 |
| $\mathbf{2^{-7}}$ | 0.0078125 |
| $\mathbf{2^{-8}}$ | 0.00390625 |
| $\mathbf{2^{-9}}$ | 0.001953125 |

En la Tabla 3 se muestra la equivalencia de un número del sistema decimal en sistema binario:

**Tabla 3.** Equivalencia de números del sistema decimal en el sistema binario

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

### 1.1. Importancia del sistema binario en el procesamiento de datos en computadoras

La representación binaria de los datos no solo es clave para guardar y enviar información digital, sino que es muy importante para mejorar el procesamiento dentro de los sistemas informáticos. Zhabin y Zhabina [3] demostraron que usar versiones del sistema binario, como los sistemas binarios redundantes que emplean los dígitos -1, 0 y 1, pueden llegar a acelerar considerablemente los cálculos en computadoras con arquitecturas paralelas.

Estas computadoras conectadas entre sí procesan los datos dígito por dígitos, comenzando por los más significativos, lo que se traduce en un menor tiempo de espera y en una mejora del rendimiento en tiempo real. Además, al trasmitir datos de forma binaria, se reduce el consumo de recursos de hardware como pines y conexiones internas [3]. Esto demuestra que el sistema binario puede ser una herramienta importante en el desarrollo de arquitecturas eficiente y que cuenten con un alto rendimiento para el procesamiento de datos.

### 1.2. Operaciones aritméticas básicas en el sistema binario

En el sistema de numeración binario se pueden realizar las operaciones aritméticas básicas siguiendo pequeñas reglas en cada operación. Estas operaciones son fundamentales en el procesamiento de datos, ya que permiten a las computadoras realizar cálculos utilizando los dígitos 0 y 1 [4].

#### 1.2.1. Suma de números binarios

La suma de números binarios es un proceso importante en la aritmética digital, está basada en un conjunto de reglas simples que se derivan del sistema binario [4], como se puede observar en la Tabla 4:

**Tabla 4.** Reglas de la suma de binarios

| Bit A | Bit B | Resultado | Carry |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Cuando sumamos 1 + 1 en binario, el resultado es 0 y se produce un acarreo o carry de 1 a la siguiente posición de mayor orden. Es similar cuando realizamos una suma en el sistema decimal que al obtener un número mayor que 9, por ejemplo 6 + 6 = 12, dejamos el 2 y llevamos el 1 a la siguiente posición. El carry es esencial en la arquitectura de los sumadores lógicos en procesadores digitales [4].

Ejemplo:

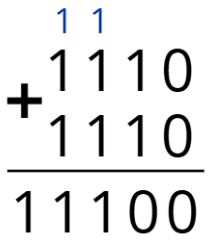


**Figura 1.** Suma de binarios

### 1.2.2. Resta de números binarios

Al igual que en la suma de binarios, la resta de binarios se basa en un conjunto específico de reglas del sistema binario [4], las cuales se pueden visualizar en la Tabla 5:

**Tabla 5.** Reglas de la resta de binarios

| Bit A | Bit B | Resultado | Borrow |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |

Cuando se realiza una resta binaria y tenemos la operación 0 – 1, se debe prestar un 1 o borrow de la siguiente posición más significativa. El borrow en binario es de suma importancia, ya que los circuitos encargados de la resta lo gestionan. Su correcta implementación es de vital importancia no solo para realizar operaciones aritméticas con números negativos, sino que también son esenciales para el diseño y funcionamiento eficiente de los microprocesadores que utilizan el complemento a dos [4].

Ejemplo:

$$\begin{array}{r} {\scriptstyle 1\phantom{111}} \\ 1\,1\,{\scriptstyle 1}0\,1 \\ -\;\;0\,0\,1\,1 \\ \hline 1\,0\,1\,0 \end{array}$$

**Figura 2.** Resta de binarios

### 1.2.3. Multiplicación de números binarios

En la multiplicación binaria, se suma el multiplicando según el valor de cada bit del multiplicador, y se aplica un desplazamiento a la izquierda por posición [4]. En la Tabla 6 se muestran sus reglas:

**Tabla 6.** Reglas de la multiplicación de binarios

| Bit A | Bit B | Producto |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

En la multiplicación de binarios no se utiliza directamente el carry o el borrow, sin embargo, los carries si son necesarios y deben ser manejados por el sistema durante la suma de los productos parciales [4].

Ejemplo:

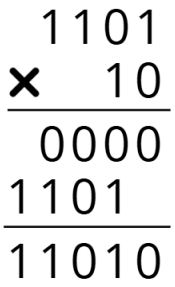


**Figura 3.** Multiplicación de binarios

### 1.2.4. División de números binarios

La división de números binarios se fundamenta en comparar segmentos del dividendo con el divisor, restar y desplazar bits [4]. En la Tabla 7 de detallan sus reglas básicas:

| Dividendo | Divisor | Cociente | Resto |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | - | - |
| 0 | 0 | - | - |

**Tabla 7.** Reglas de la división de binarios

Si el divisor es menor o igual que el conjunto de bits actual del dividendo, se anota un 1 en el cociente y se resta, caso contrario, se anota un 0 y se toma el siguiente bit del dividendo. Al restar se debe manejar el borrow cuando se lo requiera [4].

Ejemplo:

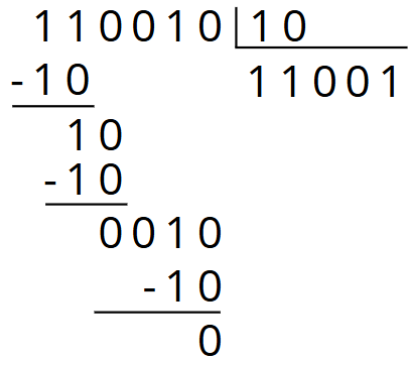


**Figura 4.** División de binarios

## 2. Operaciones en Octal y Hexadecimal

El sistema **octal** (base 8) considera los dígitos del 0 a 7 y el **hexadecimal** (base 16) usa los dígitos del 0 al 9 y las letras de la A hasta la F, que representan los valores 10 a 15 respectivamente. Estos sistemas permiten una representación compacta de los datos binarios y son habituales en la depuración, el direccionamiento de memoria, los microcontroladores y la programación de sistemas [5].

### 2.1. Conversión entre bases

**Decimal a octal o hexadecimal:** Se aplica división sucesiva por la base correspondiente, registrando los residuos de abajo hacia arriba [5].

**Entre binario y octal/hexadecimal:** Se agrupan bits en grupos de tres (base 8) o cuatro (base 16) [5].

**Por ejemplo:**

Decimal 215 → Octal: $327_8$ → 215 ÷ 8 → residuos = 3,2,7

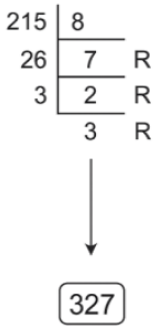


**Figura 5.** Conversión de decimal a octal

Decimal 215 → Hexadecimal: $D7_{16}$ → residuos = 13 (D), 7

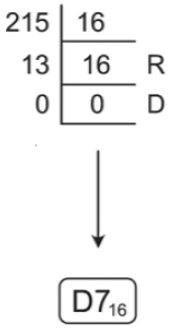


**Figura 6.** División de binarios

### 2.2. Representación digital en hardware

Los datos en las computadoras se almacenan de forma interna como combinaciones de bits, ya sea en registros o en memorias. La representación numérica en sistemas octales y hexadecimales favorece la interacción de los seres humanos con sistemas de bajo nivel. Aun así, los diagramas lógicos también dejan entrever cómo como las unidades aritmético-lógicas (ALU) de procesan tales valores [5].

## 3. Flujo de Datos dentro de una Computadora

Al referirse a la computación moderna, la base de todo esto es la arquitectura de Von Neumann. Esta se destacó al implementar en bloques separados la unidad de procesamiento y la unidad de memoria, pero a pesar de esto intercambian datos de manera intensiva y continua [6].

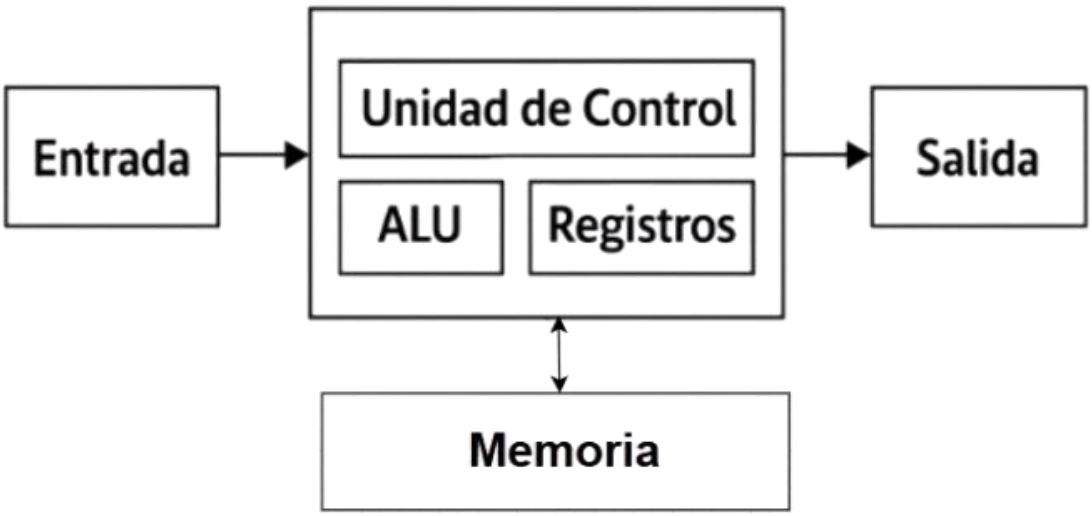


**Figura 7.** Modelo computacional de Von Neumann

En el modelo de Von Neuman para el flujo y ejecución de datos, primero las instrucciones son extraídas de la memoria, luego son decodificadas, posteriormente ejecutadas y finalmente almacenadas. Tras realizar este proceso, un contador de programa es el encargado de determinar cuál es la siguiente instrucción por ejecutar, permitiendo una ejecución de manera secuencial. Los datos sobre los que opera la instrucción u operandos son recuperados de una memoria centralizada o de los registros [6].

### 3.1. Registros

Los registros son la memoria más rápida en un computador, estos se encuentran dentro del CPU, y generalmente proporcionan una pequeña cantidad de almacenamiento. Su acceso es prácticamente inmediato debido a que se hayan en la misma CPU. Los registros son utilizados principalmente para operaciones lógicas, almacenamiento temporal de datos, contención de direcciones y operaciones a corto plazo. Son la capa principal y más rápida en la jerarquía de memorias, debido a encontrase más cerca del CPU. Entre los tipos de registros se pueden mencionar [7]:

- **Registros de Propósito General:** En arquitecturas modernas de 64 bits, por lo general suelen tener un tamaño de 64 bits. Son usados para tipos de datos integrales como lo pueden ser int y char, valores booleanos y direcciones de memoria [7].
- **Registros de Punto Flotante:** Se los utiliza para cómputo de punto flotante o vectoriales, y se los puede acceder como registros de 128 bits, dependiendo de su arquitectura y extensión. Generalmente la arquitectura A64 incluye estos tipos de registros que pueden operar sobre múltiples elementos de datos a la vez [7].
- **Registros de dispositivos de E/S:** Mediante este tipo de registros mapeados en memoria, lo controladores de dispositivos se comunican con el CPU para realizar operaciones de entrada y salida [7].
- **Registros de Sistema:** Contienen ajustes para configurar el procesador. Usados principalmente para responder y manejar excepciones [7].

### 3.2. Memoria

Todo componente físico que pueda tener datos almacenados ya sea de manera temporal o permanente se considera memoria de computadora, siendo el componente más importante. Existen varios tipos de memoria, como lo son la memoria principal, que incluyen la memoria RAM (Memoria de Acceso Aleatorio) y la ROM (Memoria Solo de Lectura). La memoria caché que puede subdividirse en caché de nivel 1 y nivel 2; y la memoria secundaria [8].

### 3.2.1. Memoria primaria

Su característica principal es su tiempo de acceso. La memoria RAM, ofrece acceso casi instantáneo a los datos a diferencia de dispositivos de almacenamiento secundario como discos duros o unidades de estado sólido (SSD). Esto debido a que es creada mediante tecnología de semiconductores y se conecta al procesador a través de un controlador de memoria. Debido a su rápida respuesta y baja latencia el procesador puede recuperar y procesar datos a un ritmo considerablemente mayor [8].

### 3.2.2. Memoria caché

También conocida como memoria de CPU, es compacta y rápida, y se encuentra cerca de la unidad central de procesamiento, (CPU) que la memoria RAM. Tiene como función acortar el tiempo que le toma al CPU almacenar datos temporalmente e instrucciones de acceso frecuente [8].

Este principio de localidad se debe a que los programas prefieren acceder a datos e instrucciones cercanos a la memoria. La caché cuenta con varios niveles, conocidos como caché L1, L2 Y L3 de los cuales cada uno es más grande que el anterior, pero a su vez más lento [8].

### 3.2.3. Memoria secundaria (almacenamiento)

Es un componente esencial de cualquier sistema informático, ya que proporciona almacenamiento a largo plazo. A diferencia de la memoria RAM que es volátil y pierde datos al interrumpirse la alimentación, esta memoria conserva datos incluso cuando el equipo se encuentra apagado. La memoria secundaria puede almacenarse en dos tipos principales [8]:

- **Magnética:** Los datos se almacenan y recuperan a través de magnetismos en dispositivos de almacenamiento magnético como lo son los discos duros (HDD), y cintas magnéticas [9]. Estos tienen gran capacidad de almacenamiento y se encuentran principalmente en computadores personales y grandes servidores. Su almacenamiento puede oscilar entre los varios gigabytes, hasta los varios terabytes [8].
- **De estado sólido:** En este tipo de almacenamiento, es usada la tecnología de memoria flash para almacenar y recuperar datos. A diferencia del almacenamiento magnético, el almacenamiento de estado sólido no contiene elementos móviles, lo que permite mejores tiempos de acceso, mayor eficiencia energética y mayor durabilidad [8].

### 3.2.4. Memoria virtual

Los sistemas operativos son aquellos que emplean memoria virtual, para aumentar la capacidad de memoria más allá de los límites de la memoria RAM física del ordenador. Está compuesta por una parte del almacenamiento secundario, y otra parte de la memoria RAM. Gracias a esta memoria, los programas pueden ejecutarse a pesar de tener poca memoria física [8].

## 3.3. Jerarquía de memorias

La jerarquía de memorias describe como se encuentran organizadas y clasificadas las distintas formas de memoria. Cada uno con velocidades y capacidades distintas, yendo desde los registros hasta el almacenamiento secundario [8].

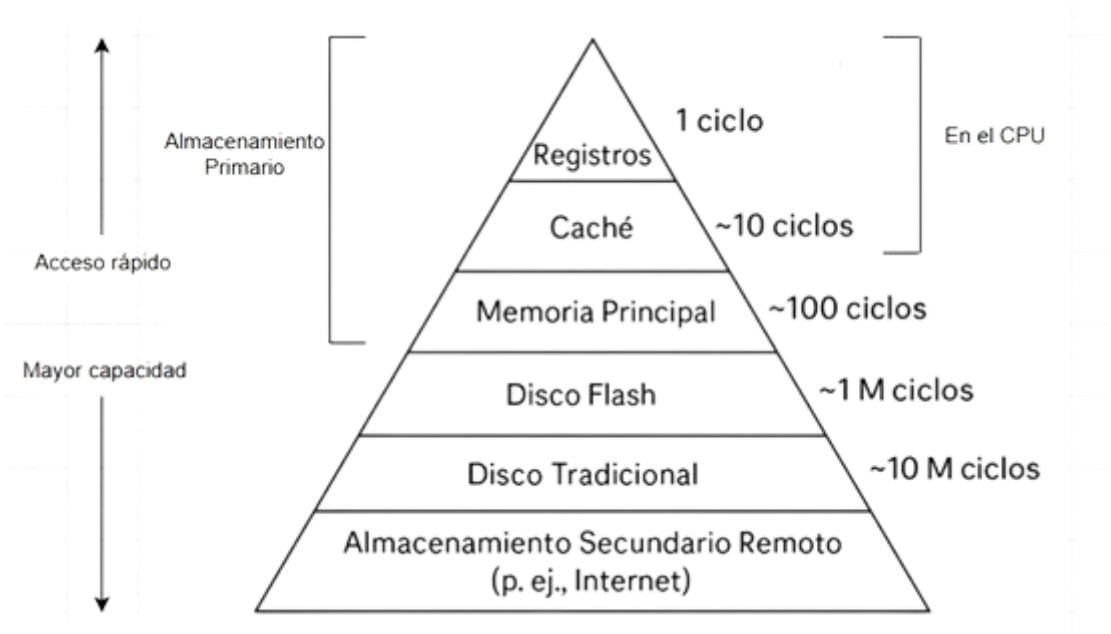


**Figura 8.** Jerarquía de la memoria en la computadora

## 3.4. Buses de sistema computacional

Un sistema computacional típico proporciona varios buses para interconectar varios componentes propios del sistema. Este satisface todas las necesidades y requisitos de todos los componentes del sistema. Sin embargo, por cuestiones como coste, retrocompatibilidad se emplean varios esquemas de interconexiones, como el que se presenta a continuación [10].

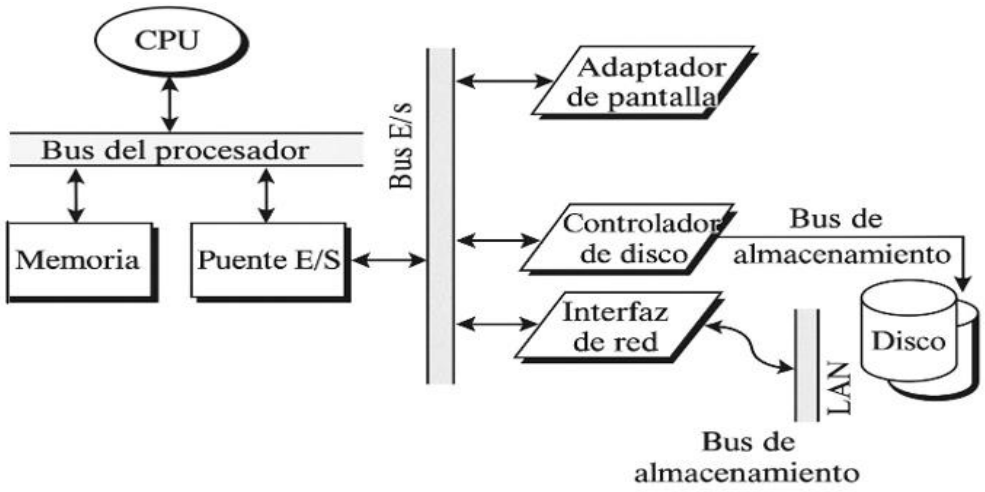


**Figura 9.** Esquema de los diferentes tipos de buses

### 3.4.1. Buses del procesador

Los buses del procesador se utilizan para conectar la CPU a la memoria principal, así como al puente E/S (Entrada y Salida). Es fundamental para el rendimiento del procesador porque esta depende de una conexión de alto ancho de banda y baja latencia con la memoria principal. Permitiendo así que las solicitudes de lectura/escritura emitidas por el CPU a la memoria, sean gestionadas por el controlador de memoria en un tiempo reducido [10].

### 3.4.2. Bus de Entrada y Salida (E/S)

Los buses E/S son utilizados para interconectar dispositivos periféricos. Como lo pueden ser interfaces de red, controladores de disco, adaptadores de pantalla, teclados o ratones, con el resto del sistema informático. Esta comunicación puede ocurrir de varias maneras [10]:

- **E/S programada:** son instrucciones específicas de la Arquitectura del Conjunto de Instrucciones (ISA), para interactuar con registros de control del dispositivo [10].
- **E/S mapeada a memoria:** los registros de control del dispositivo son asignados a direcciones en el espacio de direcciones de memoria. La CPU interactúa con los mismos utilizando instrucciones de carga y almacenamiento convencionales a través del bus. Para garantizar que la CPU vea los efectos de estas operaciones y no se queden en la caché, estas zonas suelen marcarse como caché inhibida en las tablas de páginas de memoria virtual [10].

### 3.4.3. Bus de almacenamiento

Utilizados para conectar unidades de disco magnético y otros dispositivos de almacenamiento. Proporcionan un medio para interconectar dispositivos de almacenamiento y sirven como interfaz al comunicarse. Deben ser capaces de cubrir distancias físicas significativas que la de los buses E/S, ya que los dispositivos de almacenamiento pueden encontrase en carcasas externas [10].

## 4. Códigos de Representación Numérica y No Numérica

Los sistemas digitales utilizan códigos binarios estandarizados para la representación de datos numéricos y no numéricos. Natarajan [11] explica que códigos como BCD y ASCII permiten transformar datos numéricos y alfanuméricos en secuencias binarias, adecuadas para el procesamiento por circuitos digitales. Estos códigos también simplifican el diseño de la lógica digital y permiten la interoperabilidad entre sistemas digitales y de comunicación.

### 4.1. Código ASCCI

El código ASCII (American Estándar Code for Information Interchange) es un método esencial y muy utilizado para representar caracteres como valores numéricos en sistemas computacionales. Funciona asignando un código numérico único a cada carácter permitiendo su procesamiento digital [11].

En [12], Park demuestra la rapidez con la que el código ASCI) convierte caracteres no numéricos, que luego se utilizan en el algoritmo DTW para el reconocimiento de texto en imágenes. Aunque tenga una gran velocidad y fácil implementación, puede presentar dificultades cuando los caracteres son parecidos como el 5 y la S.

En la Tabla 8 se puede visualizar el código ASCII:

**Tabla 8.** Código ASCII

| Caracteres ASCII de control | | Caracteres ASCII imprimibles | | | | | | ASCII extendido | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **00** | NULL | **32** | Espacio | **64** | @ | **96** | ` | **128** | Ç | **160** | á | **192** | └ | **224** | Ó |
| **01** | SOH | **33** | ! | **65** | A | **97** | a | **129** | ü | **161** | í | **193** | ┴ | **225** | ß |
| **02** | STX | **34** | “ | **66** | B | **98** | b | **130** | é | **162** | ó | **194** | ┬ | **226** | Ô |
| **03** | ETX | **35** | # | **67** | C | **99** | c | **131** | â | **163** | ú | **195** | ├ | **227** | Ò |
| **04** | EOT | **36** | $ | **68** | D | **100** | d | **132** | ä | **164** | ñ | **196** | ─ | **228** | õ |
| **05** | ENQ | **37** | % | **69** | E | **101** | e | **133** | à | **165** | Ñ | **197** | ┼ | **229** | Õ |
| **06** | ACK | **38** | & | **70** | F | **102** | f | **134** | å | **166** | ª | **198** | ã | **230** | µ |
| **07** | BEL | **39** | ‘ | **71** | G | **103** | g | **135** | ç | **167** | º | **199** | Ã | **231** | þ |
| **08** | BS | **40** | ( | **72** | H | **104** | h | **136** | ê | **168** | ¿ | **200** | ╚ | **232** | Þ |
| **09** | HT | **41** | ) | **73** | I | **105** | i | **137** | ë | **169** | ® | **201** | ╔ | **233** | Ú |
| **10** | LF | **42** | * | **74** | J | **106** | j | **138** | è | **170** | ¬ | **202** | ╩ | **234** | Û |
| **11** | VT | **43** | + | **75** | K | **107** | k | **139** | ï | **171** | ½ | **203** | ╦ | **235** | Ù |
| **12** | FF | **44** | , | **76** | L | **108** | l | **140** | î | **172** | ¼ | **204** | ╠ | **236** | ý |
| **13** | CR | **45** | - | **77** | M | **109** | m | **141** | ì | **173** | ¡ | **205** | ═ | **237** | Ý |
| **14** | SO | **46** | . | **78** | N | **110** | n | **142** | Ä | **174** | « | **206** | ╬ | **238** | ¯ |
| **15** | SI | **47** | / | **79** | O | **111** | o | **143** | Å | **175** | » | **207** | ¤ | **239** | ´ |
| **16** | DEL | **48** | 0 | **80** | P | **112** | p | **144** | É | **176** | ░ | **208** | ð | **240** | - |
| **17** | DC1 | **49** | 1 | **81** | Q | **113** | q | **145** | æ | **177** | ▒ | **209** | Ð | **241** | ± |
| **18** | DC2 | **50** | 2 | **82** | R | **114** | r | **146** | Æ | **178** | ▓ | **210** | Ê | **242** | ‗ |
| **19** | DC3 | **51** | 3 | **83** | S | **115** | s | **147** | ô | **179** | │ | **211** | Ë | **243** | ¾ |
| **20** | DC4 | **52** | 4 | **84** | T | **116** | t | **148** | ö | **180** | ┤ | **212** | È | **244** | ¶ |
| **21** | NAK | **53** | 5 | **85** | U | **117** | u | **149** | ò | **181** | Á | **213** | ı | **245** | § |
| **22** | SYN | **54** | 6 | **86** | V | **118** | v | **150** | û | **182** | Â | **214** | Í | **246** | ÷ |
| **23** | ETB | **55** | 7 | **87** | W | **119** | w | **151** | ù | **183** | À | **215** | Î | **247** | ¸ |
| **24** | CAN | **56** | 8 | **88** | X | **120** | x | **152** | ÿ | **184** | © | **216** | Ï | **248** | ° |
| **25** | EM | **57** | 9 | **89** | Y | **121** | y | **153** | Ö | **185** | ╣ | **217** | ┘ | **249** | ¨ |
| **26** | SUB | **58** | : | **90** | Z | **122** | z | **154** | Ü | **186** | ║ | **218** | ┌ | **250** | · |
| **27** | ESC | **59** | ; | **91** | [ | **123** | { | **155** | ø | **187** | ╗ | **219** | █ | **251** | ¹ |
| **28** | FS | **60** | < | **92** | \ | **124** | \| | **156** | £ | **188** | ╝ | **220** | ▄ | **252** | ³ |
| **29** | GS | **61** | = | **93** | ] | **125** | } | **157** | Ø | **189** | ¢ | **221** | ¦ | **253** | ² |
| **30** | RS | **62** | > | **94** | ^ | **126** | ~ | **158** | × | **190** | ¥ | **222** | Ì | **254** | ■ |
| **31** | US | **63** | ? | **95** | _ | **127** | DEL | **159** | *ƒ* | **191** | ┐ | **223** | ▀ | **255** | nbsp |

### 4.1.1. Conversión de texto a binario utilizando código ASCII

Para convertir un texto a binario separamos cada carácter y tomamos su equivalencia en el código ASCII que es un número decimal, luego este número lo convertimos a binario. ASCII utiliza 7 bits, pero a menudo se lo represente con 8 bits añadiendo un bit a la izquierda que por lo general es 0 [12].

Ejemplo:

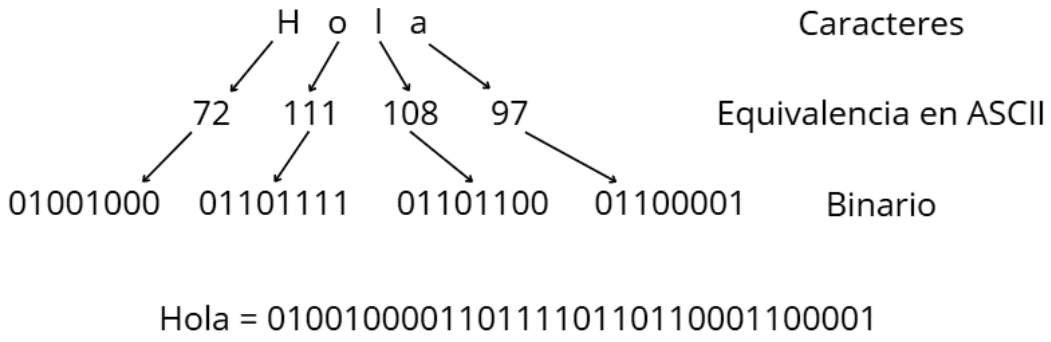


**Figura 10.** Conversión de texto a binario

### 4.1.2. Conversión de binario a texto utilizando código ASCII

Para convertir un número binario a texto, separamos en grupo de 8 bits de izquierda a derecha, en caso de que no se completen los 8 bits podemos descartar el grupo incompleto, o añadir ceros a la derecha. Luego convertimos cada grupo a sistema decimal y buscamos su equivalencia en el código ASCII [12].

Ejemplo:

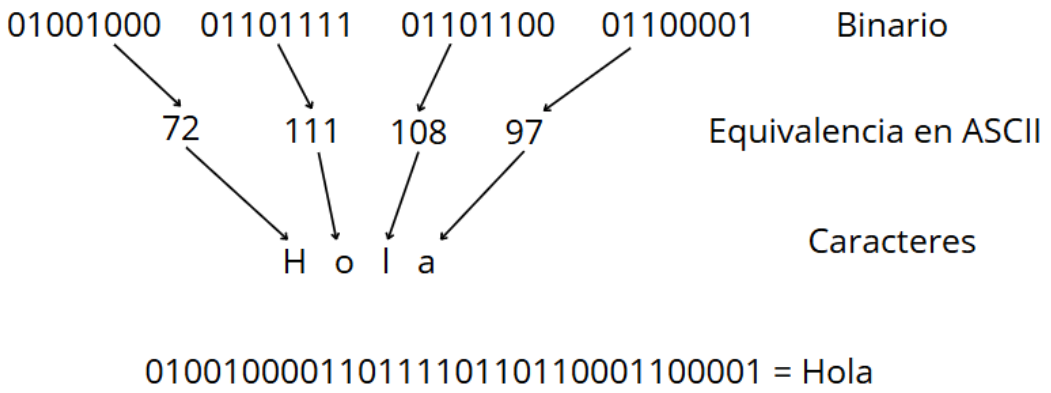


**Figura 11.** Conversión de binario a texto

## 4.2. BCD

El código BCD (Binary Coded Decimal) es un método muy utilizado para representar números decimales en formato binario. Es útil en contextos donde la exactitud decimal es esencial, como en calculadoras, relojes digitales y sistemas financieros. Este método asigna un bloque de 4 bits binarios a cada dígito decimal, del 0 al 9 [11]. Por ejemplo, 22 en BCD sería 0001 0110.

Según Natarajan [11], cada conjunto de 4 bits en BCD tiene un valor posicional fijo siguiendo el esquema 8421. Destaca que lo valores del 10 al 15 en binario no son válidos en la codificación BCD, lo que garantiza que los circuitos digitales interpreten los datos decimales de forma precisa y predecible por los circuitos digitales.

### 4.3. Códigos de corrección de errores

Los ECC permiten detectar y corregir errores sin recurrir a retransmisión, aumentando la confiabilidad en memorias volátiles (SRAM, DRAM) y no volátiles (NAND, NOR). La elección del esquema ECC depende del equilibrio entre tasa de corrección, complejidad del decodificador y eficiencia energética [13].

#### 4.3.1. Códigos superpuestos 2D

Plantean una técnica de corrección de errores mediante códigos de corrección de errores de tipo bipartito que se implementa superponiendo códigos 2D mediante la aplicación de bits de paridad a filas y columnas de una matriz de datos. Esta técnica mejora la cobertura de errores de tipo múltiple en memorias SRAM, sin necesidad de duplicar el coste en términos de la ejecución. Se presentan las métricas de latencia y overhead [13].

#### 4.3.2. Códigos Polar con decodificación avanzada

Proponen mejorar la decodificación de códigos polar sobre canales ISI bidimensionales, utilizando un decodificador SSC. Se introduce un posprocesamiento basado en perturbación aleatoria y algoritmos genéticos, que permite recuperar mensajes válidos, incluso cuando los esquemas tradicionales fallan [14].

#### 4.3.3. Códigos BCH optimizados para memorias Flash

Muestran una arquitectura optimizada para la decodificación BCH en los tipos de memoria Flash NOR y NAND. Se hace uso del paralelismo, de la técnica de *pipeline* y de multiplicadores de campo finito con compartición XOR, con el fin de que la latencia y la complejidad hardware sean reducidas al mínimo [15].

#### 4.3.4. Arquitecturas emergentes para ECC

Un análisis más reciente [16] profundiza en la comparación entre distintas arquitecturas ECC (Hamming, BCH, LDPC, Polar), evaluando el *compromiso* entre área, latencia y consumo energético. Se destaca la importancia de seleccionar arquitecturas adaptativas según el contexto (p. ej., IoT vs servidores).

### 4.4. Comparativa de esquemas ECC

**Tabla 9.** Esquemas ECC

| Código ECC | Aplicación | Ventajas principales | Limitaciones técnicas |
|---|---|---|---|
| Superpuestos 2D | SRAM y matrices de datos | Corrección cruzada en 2D con baja redundancia | Complejidad estructural creciente |
| Polar + SSC/GA | Almacenamiento 2D | Decodificación eficiente incluso bajo ISI | Alta dependencia de ajustes finos |
| BCH optimizado | Flash NAND/NOR | Alta capacidad de corrección, bajo retardo | Diseño complejo del decodificador |
| Hamming clásico | DRAM, registros simples | Eficiencia en hardware y simplicidad | Solo detecta/corrige errores simples |

## CONCLUSIÓN

Los sistemas numéricos binarios, octales y hexadecimales son la base esencial del funcionamiento de las computadoras, permitiendo operaciones aritméticas eficientes y una representación compacta de datos. Su correcta aplicación, junto con códigos como ASCII y BCD, garantiza la precisión en el procesamiento de información tanto numérica como textual. Además, la arquitectura de Von Neumann y la jerarquía de memorias demuestran cómo los datos fluyen y se gestionan de manera optimizada dentro del sistema.

**BIBLIOGRAFÍA**


[1] H. Li, "Binary System: Foundation of Modern Computing," *Mathematica Eterna*, vol. 14, no. 1, 2024, doi: 10.35248/1314-3344.24.14.206.

[2] J. Salido Tercero, *Lógica digital y tecnología de computadores. Un enfoque práctico mediante simulación con Logisim*. España: Ediciones de la Universidad de Castilla-La Mancha, 2023. doi: 10.18239/manuales_2023.26.00.

[3] V. Zhabin and V. Zhabina, "Methods of On-Line Computation Acceleration in Systems with Direct Connection between Units," in *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Kyiv: IEEE, May 2020, pp. 356–362. doi: 10.1109/DESSERT50317.2020.9125082.

[4] A. C. Jha, "Positional Number System," *NUTA Journal*, vol. 7, no. 1–2, pp. 1–9, Dec. 2020, doi: 10.3126/nutaj.v7i1-2.39924.

[5] M. Shabnam, S. Mahat, and M. K. Patil, "Number System for Digital Computers," *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH*, vol. 9, no. 4, pp. 3440–3449, Apr. 2020, [Online]. Available: https://www.ijstr.org/final-print/apr2020/Number-System-For-Digital-Computers.pdf

[6] J. Ederhion *et al.*, "Evolution, Challenges, and Optimization in Computer Architecture: The Role of Reconfigurable Systems," Dec. 2024, doi: 10.48550/arXiv.2412.19234.

[7] R. G. Plants, *INTRODUCTION TO COMPUTER ORGANIZATION: ARM*. San Francisco: William Pollock, 2025. [Online]. Available: https://dokumen.pub/introduction-to-computer-organization-arm-edition-1nbsped-1718502745-9781718502741-9781718502758-u-1769718.html

[8] V. Worlanyo Gbedawo, G. Agyeman Owusu, C. Komla Ankah, and M. Ibrahim Daabo, "An Overview of Computer Memory Systems and Emerging Trends," *American Journal of Electrical and Computer Engineering*, Oct. 2023, doi: 10.11648/j.ajece.20230702.11.

[9] A. Ranjan, A. Raha, V. Raghunathan, and A. Raghunathan, "Approximate Memory Compression," *IEEE Trans Very Large Scale Integr VLSI Syst*, vol. 28, no. 4, pp. 980–991, Apr. 2020, doi: 10.1109/TVLSI.2020.2970041.

[10] T. E. Carlson, “Bus and Memory Architectures,” in *Handbook of Computer Architecture*, 1st ed., Singapore: Springer Nature Singapore, 2024, ch. 6, pp. 201–212. doi: 10.1007/978-981-97-9314-3_68.

[11] D. Natarajan, “Number Systems and Binary Codes,” in *Fundamentals of Digital Electronics*, vol. 623, Bengaluru: Springer International Publishing, 2020, ch. 5, pp. 95–119. doi: 10.1007/978-3-030-36196-9.

[12] H. J. Park, “A method to convert non-numeric characters into numerical values in dynamic time warping for string matching,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, no. 3, pp. 2660–2665, Jun. 2021, doi: 10.11591/ijece.v11i3.pp2660-2665.

[13] A. R. Fritsch, “OVERLAPPING ERROR CORRECTION CODES ON TWODIMENSIONAL STRUCTURES,” PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL, Porto Alegre, 2025. doi: 10.48550/arXiv.2504.12142.

[14] N. K. Gerrar, S. Zhao, and L. Kong, “Error correction in data storage systems using polar codes,” *IET Communications*, vol. 15, no. 14, pp. 1859–1868, Aug. 2021, doi: 10.1049/cmu2.12197.

[15] S. Nabipour and J. Javidan, “Enhancing Data Storage Reliability and Error Correction in Multilevel NOR and NAND Flash Memories through Optimal Design of BCH Codes,” Iran, Jul. 2023. doi: 10.48550/arXiv.2307.08084.

[16] L. Parrini *et al.*, “Error Detection and Correction Codes for Safe In-Memory Computations,” *IEEE European Test Symposium*, Apr. 2024, doi: 10.48550/arXiv.2404.09818.

**ANEXO**

**Link del repositorio en GitHub**

**https://github.com/AndyMendoza0308/ArqComp-Grupo-B---Sistemas-num-ricos.git**