UNIVERSIDAD TÉCNICA ESTATAL DE QUEVEDO

FACULTAD CIENCIAS DE LA COMPUTACIÓN Y DISEÑO DIGITAL

INGENIERÍA EN SOFTWARE

MATERIA:

ARQUITECTURA DE COMPUTADORAS

TEMA:

SISTEMAS NUMÉRICOS Y REPRESENTACIÓN DE DATOS INTERNA DE LA COMPUTADORA

INTEGRANTES:

GUTIERREZ ORTEGA GENESIS ADRIANA

PEREZ RUIZ CARLOS ANDRES

PONCE RIVERA MERY HELENMEY

SABANDO TOAQUIZA INGRID ANDREINA

DOCENTE:

ING. GUERRERO ULLOA GLEISTON CICERON

CURSO/PARALELO:

SEGUNDO SOFTWARE “B”

SPA 2025 – 2025

1. OPERACIONES FUNDAMENTALES EN BINARIO

El sistema de numeración binaria consiste en representación de dos dígitos, 0 o 1, también conocidos como bits. Se puede interpretar todo tipo de información desde números, imágenes, videos y audios [1].

1.1. Importancia En El Procesamiento De Datos

La representación física de un bit en la memoria de una computadora es realizada por medio de un transistor y un condensador, (1) para estar encendido o (0) para apagado. Dentro de los discos magnéticos los bits son representados por medio de la dirección de un campo magnético y los discos ópticos o datos digitales se representan con la reflexión de la luz para (1) o no (0) [1].

Los numero binarios representan las imágenes dentro de una computadora mediante pixeles en puntos blancos, negros o en grupos para dar mostrar el color. Además, es fundamental dentro de la computación, ya sea como técnicas de comprensión, codificación y medición de información [1].

1.2. Operaciones Aritméticas Básicas

En la Unidad Aritmética Lógica (ALU) componente fundamental de la CPU encargado de realizar las operaciones aritméticas [2].

1.2.1. Suma Binaria

Dentro de la suma binaria, se ejecuta mediante circuitos especializados que implementa la lógica de la adición. El circuito básico es el semi-sumidor el cual, se encarga de sumar dos dígitos binarios generando una salida de un bit de suma (S) y un bit de acarreo (C).

La salida de la suma corresponde a una operación OR-Exclusiva, mientras que, el de acarreo da una salida de operación AND-Lógica y este resulta en 1 solo cuando ambas entradas son 1 [2].

El manejo del acarreo es esencial y es manejado mediante de redes de propagación de acarreo, que generan señales auxiliares para indicar cuando un acarreo es dado $(gi=xiyi)$ o cuando se propaga $(pi=xi \oplus yi)$ y esta gestión da lugar a distintas arquitecturas de sumadores [2].

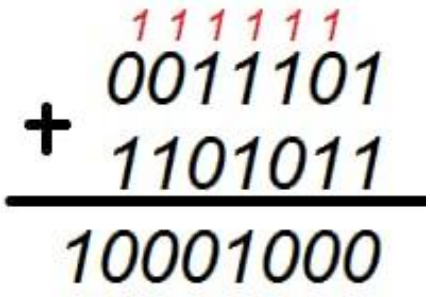

Figura 1: Suma binaria

1.2.2. Resta Binaria

La resta es otra de las operaciones aritméticas básicas que es ejecutada por la Unidad Aritmética Lógica (ALU), la resta binaria se realiza principalmente mediante el método de complemento a dos. Un enfoque fundamental en el diseño de procesadores, el cual permite realizar sustracciones utilizando los circuitos de la suma [2].

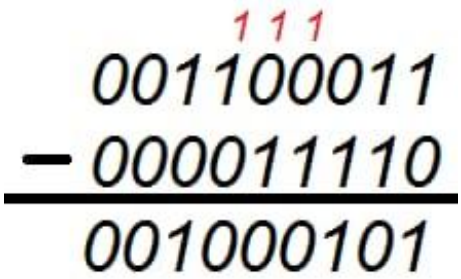

Figura 2: Resta Binaria

1.2.3. Multiplicación Binaria

La multiplicación binaria es considerada una operación más compleja a diferencia de la suma, ya que, requiere la implementación de algoritmos y circuitos especiales dentro del procesador, la multiplicación es producida si el producto de dos números enteros devuelve un resultado mayor y durante la operación, puede producirse un desbordamiento si el producto resulta mayor al valor máximo representable por el número de bits del sistema [2].

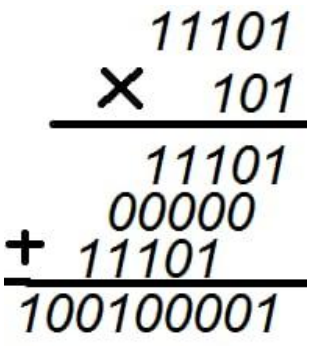

Figura 3: Multiplicación Binaria

1.2.4. División Binaria

Al igual que la multiplicación la división también depende de algoritmos y circuitos especiales diseñados para este propósito dentro del procesador, también, considerándose una operación compleja para su ejecución a nivel hardware [2].

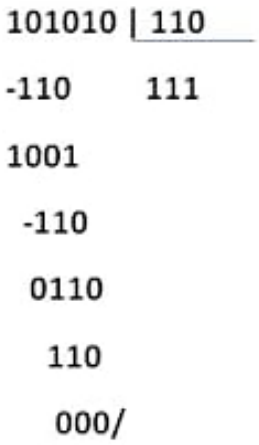

Figura 4: División Binaria

2. OPERACIONES EN OCTAL Y HEXADECIMAL

En el sistema octal, cada número está compuesto por dígitos que van del 0 al 7, y cada posición representa una potencia de 8. Esta forma de numeración simplifica la representación de valores binarios, ya que tres bits binarios corresponden exactamente a un dígito octal. De manera similar, el sistema hexadecimal utiliza dígitos del 0 al 9 y letras de la A a la F, donde cada posición es una potencia de 16, y cada dígito hexadecimal equivale a cuatro bits binarios [3].

Tanto el sistema octal como el hexadecimal se usan porque permiten representar números binarios largos de forma más compacta y legible, lo que es útil en programación y hardware.

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

Figura 5: Sistema Octal

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

Figura 6: Sistema Hexadecimal

2.1.Utilidad Del Sistema Hexadecimal En Programación

El sistema hexadecimal se utiliza frecuentemente para representar instrucciones en lenguaje ensamblador. Gracias a que cada instrucción máquina está formada por múltiples bits, escribirlas en binario sería muy largo y poco práctico. En cambio, expresarlas en hexadecimal permite comprimir esa información y trabajar de forma más eficiente en entornos de bajo nivel [4].

2.2. Hexadecimal Y La Relación Con Los Bytes

Una ventaja adicional del sistema hexadecimal es su alineación natural con los bytes, ya que un byte (8 bits) puede representarse como dos dígitos hexadecimales. Esto hace que el trabajo con direcciones de memoria y valores almacenados en registros sea más intuitivo. Por eso, muchos lenguajes y depuradores muestran valores de memoria en hexadecimal: es una forma directa de ver el contenido real de un byte [4].

Entender que "A1" equivale a un byte específico ayuda a interpretar qué instrucciones o datos están siendo manejados en una dirección concreta de memoria.

2.3. Operaciones Básicas En Octal Y Hexadecimal

En operaciones aritméticas, tanto en octal como en hexadecimal, las reglas básicas se adaptan al valor de su base. Por ejemplo, en octal, si al sumar dos dígitos se pasa de 7, se lleva una unidad a la siguiente posición. En hexadecimal ocurre lo mismo si se supera la F (que representa el 15 decimal) [3].

En la figura 7 se muestra un ejemplo de suma octal. Aquí se observa cómo al sumar dos dígitos que dan un resultado mayor a 7, se lleva 1 y se ajusta el resultado a base 8.

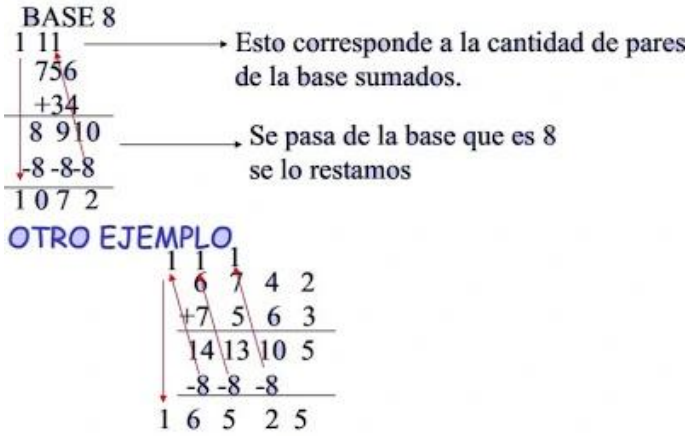

Figura 7: Suma Octal

En la figura 8 se muestra un ejemplo de resta octal. Se puede ver que cuando no se puede restar directamente, se pide prestado del dígito de la izquierda, teniendo en cuenta la base octal.

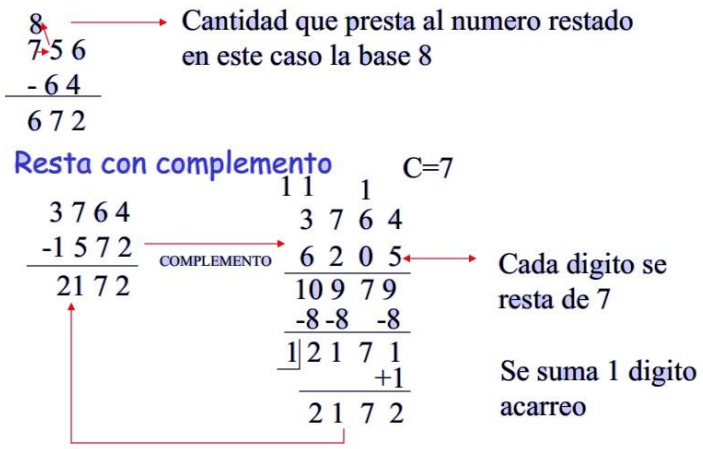

Figura 8: Resta Octal

En la figura 9 se muestra un ejemplo de suma hexadecimal. Se puede ver que cuando el resultado supera F (15), se genera un acarreo, y se ajusta el valor al sistema hexadecimal.

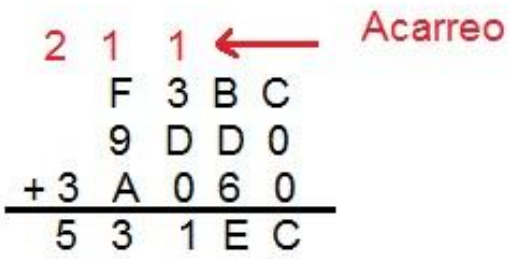

Figura 9: Suma Hexadecimal

En la figura 10 se muestra un ejemplo de resta hexadecimal. En este caso, la operación se puede realizar directamente sin necesidad de pedir prestado, simplemente restando los valores correspondientes en hexadecimal.

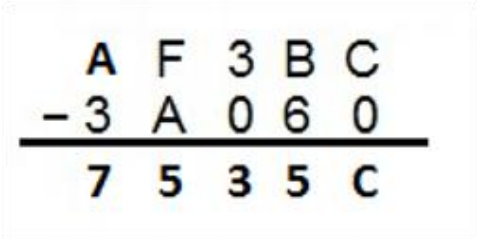

Figura 10: Resta Hexadecimal

Además de facilitar la representación binaria, los sistemas octal y hexadecimal permiten transformar cifras complejas a otras bases de forma directa y estructurada. Por ejemplo, al convertir de hexadecimal a binario, se toma cada dígito y se traduce directamente en un grupo de cuatro bits, mientras que en el caso del octal se usa un grupo de tres bits.

2.4. Aplicación Práctica En Arquitectura De Computadores

En la arquitectura de computadores, el uso de números en base 8 y 16 es esencial para la representación de instrucciones máquina, direcciones de memoria y manipulación de registros. Al utilizar sistemas como el hexadecimal, se puede comprender más fácilmente el comportamiento del procesador y los datos almacenados. Este sistema actúa como una especie de "puente" entre el binario que entiende la máquina y el ensamblador que programa el humano [4].

3. FLUJO DE DATOS DENTRO DE UNA COMPUTADORA

El flujo de datos en una computadora comienza con la interacción entre la CPU y la memoria, utilizando registros como el MAR, que señala direcciones, y el MDR, que transporta datos. La memoria RAM, que es volátil, y la ROM, que es no volátil, cumplen funciones específicas: la RAM permite un acceso rápido y temporal, mientras que la ROM se encarga de almacenar datos de manera permanente. El bus de datos facilita esta comunicación, que está organizada de manera jerárquica para optimizar el rendimiento[5].

Durante la fase de Fetch, el Program Counter (PC) señala la dirección de la instrucción que se va a recuperar de la memoria. Una vez que se obtiene (32 bits), el PC se incrementa automáticamente en 4 unidades (PC + 4), a menos que una instrucción de salto cambie esta secuencia a través de un multiplexor que elige la nueva dirección. En esta etapa, solo se utiliza el registro PC [6].

En la fase de Decode, la instrucción se interpreta a través de los campos OpCode, Funct3 y Funct7, que activan señales específicas producidas por la Unidad de Control. Se accede a los registros fuente (rs1 y rs2) y se calcula un valor inmediato utilizando el Generador Inmediato, extendiendo su signo para crear un dato de 32 bits que se utilizará en la siguiente etapa [6].

La ALU, o Unidad Aritmética Lógica, se encarga de procesar los operandos que recibe de los registros. Realiza diversas operaciones, ya sean aritméticas como la suma o la resta,

o lógicas como AND, OR y XOR, además de manejar corrimientos de bits, todo dependiendo del código de operación que le llega (ALUOp). Al final, el resultado se presenta en una salida de 32 bits, completando así el ciclo de ejecución de la instrucción [6].

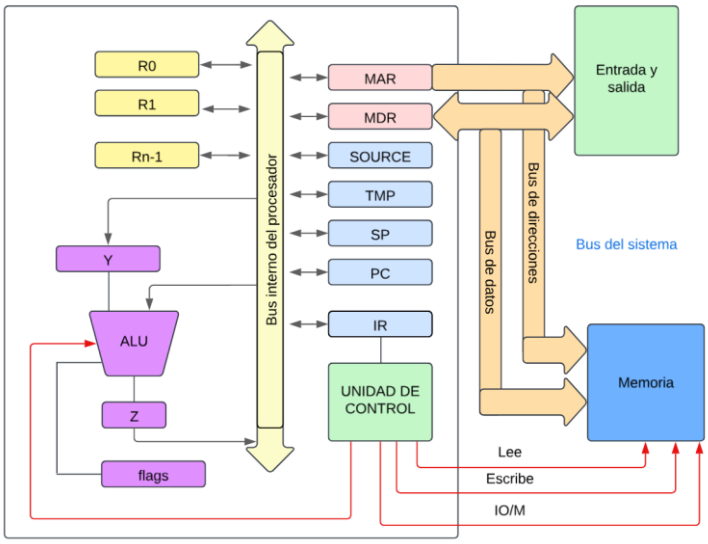

Figura 11: Flujo de Datos

4. CÓDIGOS DE REPRESENTACIÓN NUMÉRICA Y NO NUMÉRICA

En esta sección se explican los métodos usados para representar números y caracteres en sistemas digitales, esenciales para procesar datos de forma eficiente y precisa.

4.1 CÓDIGOS DE REPRESENTACIÓN NUMÉRICA

Los códigos de representación numérica son esquemas que representan datos cuantitativos (enteros o reales) en formatos binarios optimizados para operaciones eficientes. Un ejemplo destacado es la representación signed – digit, en la cual cada dígito puede ser positivo o negativo. Esto permite realizar operaciones, como la suma; en el tiempo constante sin propagación de acarreo, lo cual es crucial en procesadores de alto rendimiento y hardware especializado [7].

A continuación, se verán tipos de códigos de representación numérica:

4.1.1. Representación En Aritmética Paralela (Signed-Digit)

La representación signet-digit (dígito con signo) es una técnica moderna donde cada posición puede tomar valores positivos o negativos, eliminando así la necesidad de

propagar el acarreo entre dígitos. Esto permite ejecutar sumar y restas en tiempo constante, una característica clave en procesadores paralelos y hardware especializado. Su aplicación en sistemas embebidos y de procesamiento de señales es fundamental para optimizar el uso del hardware [8].

4.1.2. Convenciones De Representación Numérica (Overflow)

Las convenciones de representación numérica definen como se gestionan los errores cuando los valores calculados exceden el rango permitido por el sistema. Esto se conoce como overflow, y su detección y manejo son fundamentales para preservar la integridad de los resultados. En arquitecturas modernas, estas convenciones se implementan desde el diseño del hardware o mediante lenguajes formales que detectan y validan limites numéricos, evitando fallos en tiempo de ejecución [9].

4.1.3. La Codificación Universal De Números Reales

La codificación universal de números reales consiste en representar valores reales como secuencias binarias finitas o infinitas mediante técnicas como bisección de intervalos. Esta estrategia permite representar valores continuos con un alto grado de precisión; superando las limitaciones de formatos tradicionales. Es especialmente útil en aplicaciones cientifícas y criptográficas donde se requiere trabajar con precisión arbitraria y sin errores de redondeo acumulativo [10].

4.2. CÓDIGOS DE REPRESENTACIÓN NO NUMÉRICA

Los códigos de representación no numérica engloban formatos para codificar elementos como texto, símbolos y caracteres Unicode en sistemas informáticos. Un ejemplo reciente es la validación eficiente de texto CESU-8 mediante instrucciones SIMB. Esta técnica permite procesar y validar bloques de texto multilingüe de forma paralela, garantizando precisión y compatibilidad en aplicaciones modernas [7].

A continuación, se verán tipos de códigos de representación no numérica:

4.2.1. Extensión ASCII Para Texto

El código ASCII, aunque ampliamente adoptado, fue originalmente limitado a 128 caracteres. Ante la necesidad de incluir alfabetos no latinos, símbolos matemáticos y caracteres gráficos, se desarrollaron múltiples extensiones compatibles con ASCII. Una de las más destacadas fue diseñada para permitir el uso de símbolos no numéricos en

ambientes interactivos, permitiendo que los sistemas pudieran operar sin errores al alternar entre diferentes conjuntos de caracteres [12].

Este tipo de codificación se volvió vital para sistemas operativos, terminales, procesadores de texto y navegadores web, especialmente en un mundo cada vez más globalizado y multilingüe. La compatibilidad hacia atrás con ASCII aseguraba que los documentos antiguos pudieran seguir siendo interpretados, mientras que la extensión proporcionaba mayor expresividad [12].

4.2.2. IPC: Information Processing Code

El IPC fue diseñado como una solución avanzada para el procesamiento de datos no numéricos, especialmente texto. Se trata de un código de 8 bits, ampliable a 9 bits, que distribuye sus valores entre caracteres alfabéticos, de control y otros símbolos necesarios en el procesamiento de documentos, edición textual y recuperación de información. Una de sus características más destacadas es que conserva compatibilidad parcial con ASCII, pero con una extensión que facilita la interacción entre sistemas con diferentes alfabetos y funciones especiales [13].

Esta codificación fue fundamental en los primeros sistemas interactivos y sigue teniendo relevancia histórica, al haber sentado las bases de posteriores estándares de codificación multilingüe. Gracias a su diseño modular, permitió expandir el conjunto de caracteres sin alterar la lógica central de los sistemas que lo empleaban [13].

- Conversión de decimal a binario y viceversa

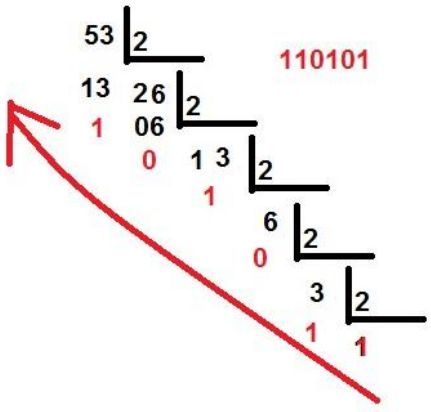

Figura 12: Decimal a Binario

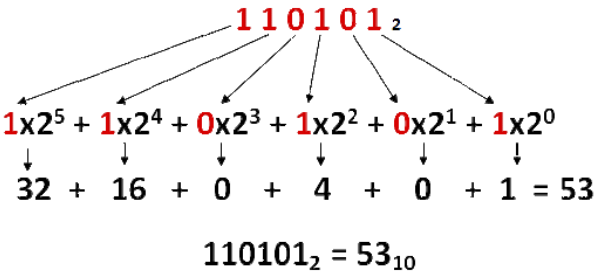

Figura 13: Binario a Decimal

BIBLIOGRAFÍA

[1] F. Coelho, G. Gonçalves, W. Massashiro, and M. Aurélio Alvarenga, "PENSAMENTO COMPUTACIONAL: PROPOSTA METODOLÓGICA PARA O ENSINO DE NÚMEROS BINÁRIOS COMPUTATIONAL THINKING: METHODOLOGICAL PROPOSAL FOR THE TEACHING OF BINARY NUMBERS," no. 1, pp. 129–153, 2021.

[2] D. P. López Carrillo, Á. M. Acuña Félix, R. F. Tipan Tisalema, G. I. Vanegas Zabala, and D. F. Yumisa León, *Arquitectura de computadoras*. Centro de Investigación y Desarrollo Ecuador, 2025. doi: 10.33996/cide.ecuador.AC2679376.

[3] 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.

[4] S. B. Mir *et al.*, "Introducción a la arquitectura de computadores con QtARMSim y Arduino," 2019. Accessed: Jun. 07, 2025. [Online]. Available: https://dspace.itsjapon.edu.ec/jspui/bitstream/123456789/435/1/Introduccion-ARM-Arduino.pdf

[5] N. Pérez Ayup, *Arquitectura de computadoras. Las memorias en la PC*. 2020.

[6] J. A. Jaramillo Villegas, H. M. Zuluaga Bucheli, and C. Sepúlveda Caviedes, *Arquitectura de Computadoras con RISC-V*. Universidad Tecnológica de Pereira, 2022. doi: 10.22517/9789587227956.

[7] L. Monroe, "Binary signed-digit integers and the Stern diatomic sequence," *Des Codes Cryptogr*, vol. 89, no. 12, pp. 2653–2662, Dec. 2021, doi: 10.1007/S10623-021-00903-6.

[8] L. Monroe, "Optimal binary signed-digit representations of integers and the Stern polynomial," *Des Codes Cryptogr*, vol. 92, no. 6, pp. 1501–1516, Jun. 2024, doi: 10.1007/S10623-023-01355-W/METRICS.

[9] M. Boreale, "Complete algorithms for algebraic strongest postconditions and weakest preconditions in polynomial odes," *Sci Comput Program*, vol. 193, p. 102441, Jul. 2020, doi: 10.1016/J.SCICO.2020.102441.

[10] M. Navarro, F. Orejas, E. Pino, and L. Lambers, “A navigational logic for reasoning about graph properties,” *Journal of Logical and Algebraic Methods in Programming*, vol. 118, p. 100616, Jan. 2021, doi: 10.1016/J.JLAMP.2020.100616.

[11] M. Schröder, S. MacHmeier, S. Maeng, and V. Heuveline, “Validating CESU-8 Encoded Text Utilising SIMD Instructions,” *ACM International Conference Proceeding Series*, pp. 102–111, Feb. 2024, doi: 10.1145/3651781.3651797/SUPPL_FILE/ICSCA_SLIDES_POST.PDF.

[12] S. Gorn, “Code extension in ASCII,” *Commun ACM*, vol. 9, no. 10, pp. 758–762, Oct. 1966, doi: 10.1145/365844.365870;GROUPTOPIC:TOPIC:ACM-PUBTYPE>MAGAZINE;WGROUP:STRING:ACM.

[13] E. Morenoff and J. B. McLean, “A code for non-numeric information processing applications in online systems,” *Commun ACM*, vol. 10, no. 1, pp. 19–22, Jan. 1967, doi: 10.1145/363018.363042.

ANEXOS

Link de github

https://github.com/GenessiGutierrez/SISTEMAS-N-MERICOS---GRUPO-D/tree/main