Organización de Computadores

Es el estudio de las unidades funcionales y sus interconexiones, que dan lugar a especificaciones arquitectónicas; conjunto de instrucciones, número de bits para representar tipos de datos, mecanismos de E/S y técnicas de direccionamiento de memoria.

- **Secuencia de Contenidos:** 2

Haciendo clik encima del número del índice se dirige  directamente al tema

3. Sistema de Numeración:

Un **sistema de numeración** es un conjunto de símbolos o numerales y reglas de generación que permiten construir todos los números válidos en el sistema.

Un sistema de numeración puede representarse como:

- $N = (S, R)$

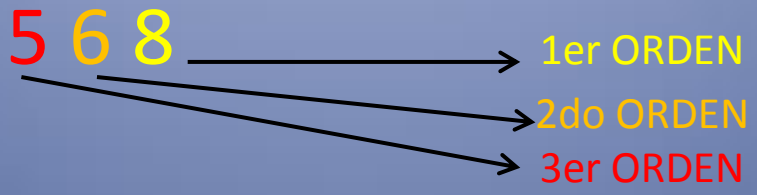
Donde:

- N es el sistema de numeración considerado, por ejemplo: decimal, binario, etc.
- S es el conjunto de símbolos permitidos en el sistema. En el caso del sistema decimal son: 0,1,2,3, 4, 5,,6,7,8,9; en el caso binario son: 0, 1; en el octal son: 0,1,2,3, 4, 5,,6,7; en el hexadecimal: 0,1,2,3, 4, 5,,6,7,8,9, A,B,C,D,E,F, …etc.
- R son las reglas que nos indican que números son válidos en el sistema y cuales no. En un sistema posicional las reglas son bastante simples, mientras que en la numeración romana requiere algo más elaboradas.

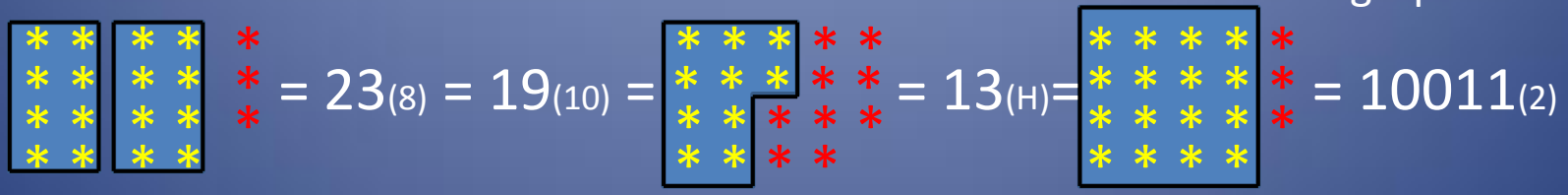
Estas reglas son diferentes para cada sistema de numeración considerado, pero una regla común a todos es que para construir números válidos en un sistema determinado solo se pueden utilizar los símbolos permitidos en dicho sistema.

3. Sistema de Numeración:

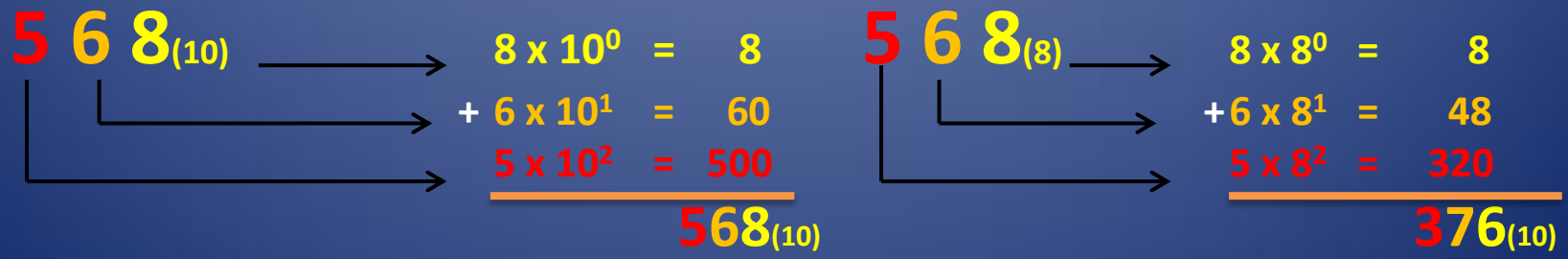
- **REGLAS o PRINCIPIOS:** Estas son establecidas para diferenciar cada sistema numérico.
- **NUMERALES:** Los símbolos o numerales un conjunto pueden pertenecer aotro sistema.
- **NIDENTIFICACIÓN:** Un sistema numérico se identifica por el subíndice a la derecha.
- **PRINCIPIO de ORDEN:** Toda cifra de un numeral tiene un orden de jerarquía :



- **PRINCIPIO de BASE:** Todo sistema de numeración nos alude a como agrupar.



- **PRINCIPIO POSICIONAL:** Toda cifra de un numeral tiene un orden de jerarquía :



3. Sistema de Numeración:

TABLA de CORRESPONDENCIA: A todas las series numéricas se les asigna un cero como punto de referencia o inicio, siendo el propio cero un elemento de la serie, cuyo valor siempre estará determinado por la posición en la cifra.

EL NÚMERO: El número se representa mediante una secuencia de cifras : $N=n_n+n_{n-1}+n_{n-2}+\dots+n_{n2}+n_{n1}+n_{n0}$

CIFRA SIGNIFICATIVA: Se llama cifra significativa a toda cifra diferente a **CERO**.

VALOR NUMÉRICO: El valor numérico o rango de representación, se obtiene mediante el cálculo del polinomio de su representación, respetando el principio posicional del numeral.

$$\sum_{i} n_i b^i = n_n b^n+n_{n-1}b^{n-1}+n_{n-2}b^{n-2}+\dots+n_2 b^2+n_1 b^1+n_0 b^0$$

I = es el rango , ej.: -3 < i < 4
b = es la base y n = numeral

TABLA de CORRESPONDENCIA

DECIMAL	HAXADECIMAL	OCTAL	BINARIO
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111
16	10	20	10000

VOLVER AL INDICE

3.1. Sistemas de Representación:

SISTEMA BINARIO o de BASE 2: Referido a la electrónica, más específicamente a la computación, entenderemos que se trata de la apertura o cierre de compuertas en el fluido electrónico. Cero (0) será entonces el no pasaje de electricidad* y uno (1) el pasaje eléctrico.

Para su representación utilizaremos dos dígitos (0, 1) binarios, de tal modo que en la ecuación $\Sigma n_i b^i$, **n** representara tanto 1 como 0, mientras que **b** será 2 a la potencia de la posición en el numeral.

ej.: **1010.011 (2) = 10.375 (10)**

$$\Sigma n_i b^i = 1*2^3+0*2^2+1*2^1+0*2^0.0*2^{-1}+1*2^{-2}+1*2^{-3} =$$

$$-3 \leq i \leq 4$$

$$\Sigma n_i b^i = 8 + 0 + 2 + 0 . 0 + 0.25+0.125 = 10.375 \text{ (10)}$$

$$-3 \leq i \leq 4$$

$1010.011_{(2)} = 1*2^3+0*2^2+1*2^1+0*2^0.0*2^{-1}+1*2^{-2}+1*2^{-3} = 8 + 0 + 2 + 0 . 0 + 0.25+0.125 = 10.375_{(10)}$

$110100_2 = (1\cdot 2^5) + (1\cdot 2^4) + (1\cdot 2^2) = 2^5 + 2^4 + 2^2 = 32 + 16 + 4 = 52_{10}$

$0{,}10100_2 = 2^{-1} + 2^{-3} = (1/2) + (1/8) = 0{,}5 + 0{,}125 = 0{,}625_{10}$

VOLVER AL INDICE

3.1.1. Arimétrrica Binaria:

SUMA BINARIA: La suma binaria al igual que la suma decimal o de otra base, suma los dígitos ordenadamente y por orden exponencial de la base, acarreando el exceso al exponencial inmediato siguiente.

RESTA BINARIA: La resta binaria al igual que la resta decimal o de otra base, resta los dígitos ordenadamente y por orden exponencial de la base, acarreando el déficit del exponencial inmediato mayor.

Multiplicación y División BINARIA: Al igual que las anteriores se opera de la misma forma. Veamos los ejemplos:

SUMA

```
 1110101
+
 1110110
---------
11101011
```

RESTA

```
 1101010
_
 1010011
---------
 0010011
```

Multiplicación

```
 1101010
 X
      11
---------
 1101010
1101010
---------
100111110
```

División

```
1101011 | 101
-101    |-----
 00110  | 10.101
  -101
  00111
    -101
      10
```

SUMA		
A	B	A + B
0	0	o
0	1	1
1	0	1
1	1	0 (1)

RESTA		
A	B	A - B
0	0	0
0	1	1 (1)
1	0	1
1	1	0

MULTIPLICACIÓN		
A	B	A * B
0	0	0
0	1	0
1	0	0
1	1	1

DIVISIÓN		
A	B	A / B
0	0	___
0	1	0
1	0	___
1	1	1

VOLVER AL INDICE

3.2. Sistemas de Representación:

SISTEMA OCTAL o de BASE 8: Referido a la electrónica, es más comparable con el sistema de vías o de buses, por su similitud aplicada al Byte y su codificación numérica, pero menos útil que el Hexadecimal.

Para su representación utilizaremos los dígitos del 0 al 7, de tal modo que en la ecuación $\Sigma n_i b^i$, **n** representará cualquiera de los dígitos comprendidos, mientras que **b** será 8 a la potencia de la posición en el numeral.

ej.: **4 5 7 . 5 (8) = 3 0 3. 6 2 5 (10)**

$$\Sigma n_i b^i = 4*8^2+5*8^1+7*8^0+5*8^{-1} =$$
$$-3 \le i \le 4$$

$$\Sigma n_i b^i = 256 + 40 + 7 + 0.625 = 303.625_{(10)}$$
$$0 \le i \le 2$$

$$4\,5\,7_{(8)} = 4 * 8^2 + 5 * 8^1 + 7 * 8^0 + 5 * 8^{-1} = 256 + 40 + 7 + 5 * 0.125 = 303.625_{(10)}$$

$8^{-1} = 0.125_{10}$

$8^{-2} = 0.125 / 8 = 0.015625_{10}$

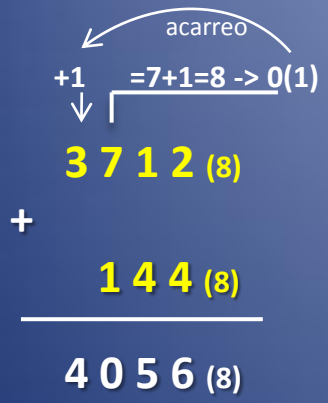
$8^{-3} = 0,125 / 8 / 8 = 0,001953125_{10}$

VOLVER AL INDICE

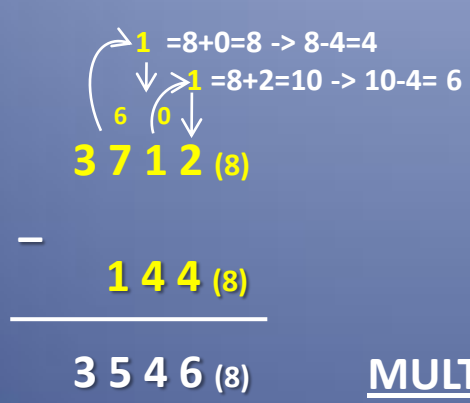
3. 2.1. Aritmética Octal:

En la aritmética Octal(8)(q) ó utilizamos los mismos recursos y a la misma lógica de la aritmética decimal, recurrimos al acarreo de unidades a la posición del exponente siguiente, le sustrae al dígito del exponente inmediato superior, utilizamos la propiedad distributiva y la suma en la multiplicación; y por, en la división usamos el procedimiento de cascada.

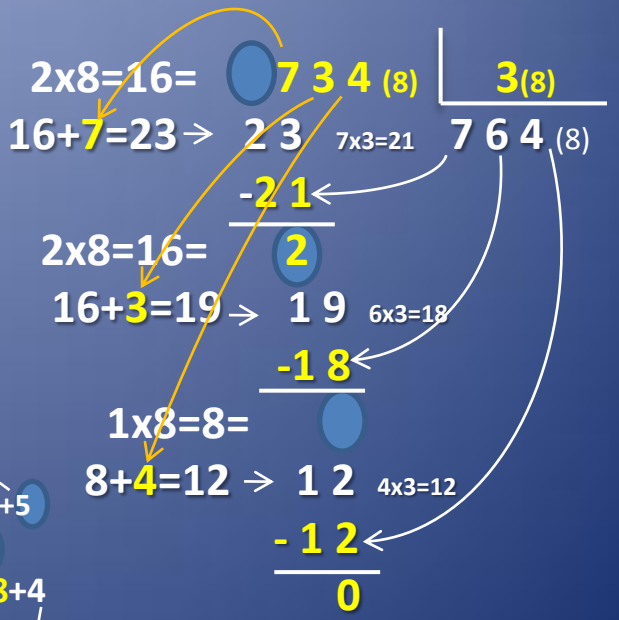
SUMA



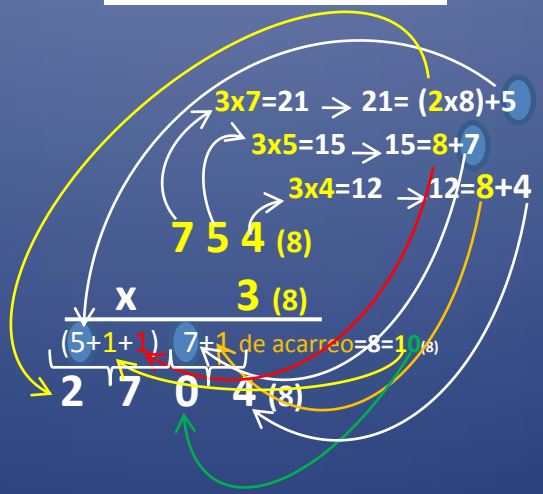
RESTA



DIVISIÓN



MULTIPLICACIÓN



VOLVER AL INDICE

3.3. Sistemas de Representación:

SISTEMA HEXADECIMAL o de BASE 16: En electrónica, por su multiplicidad aplicada al número de bits en relación al Byte se utiliza en buses multidireccionales,, pero de notación más compleja que el Octal.

Para su representación numérica utilizaremos los dígitos del 0 al 9 y las letras de la 'A 'a la 'F' para designar los valores del 10 al 16, de tal modo que en la ecuación $\Sigma n_i b^i$, **n** representará cualquiera de los dígitos comprendidos, mientras que **b** será 16 a la potencia de la posición en el numeral.

ej.: **2 5 D F . B A (16) = 9 6 9 5 . 7265 (10)**

$$\Sigma_{-3 \le i \le 4} n_i b^i = 2*16^3 + 5*16^2 + D*16^1 + F*16^0 + B*16^{-1} + A*16^{-2} =$$

$$\Sigma_{0 \le i \le 2} n_i b^i = 8192 + 1280 + 280 + 15 + 0.6875 + 0.039 = 9\,6\,9\,5\,.\,7265\ (10)$$

$25DF_{(H)} = 2*16^3 + 5*16^2 + D*16^1 + F*16^0 + B*16^{-1} + A*16^{-2} = 8192 + 1280 + 208 + 15 + 0.6875 + 0.039 =$
$25DF_{(H)} = 9695.7265\ (10)$

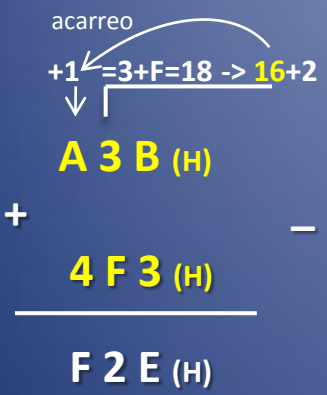
$16^{-1} = 0.0625_{10}$

$16^{-2} = 0.0625 / 16 = 0.00390625_{10}$

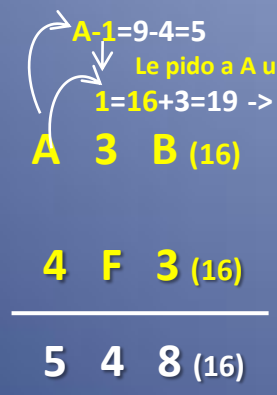
3.3.1. Aritmética Hexadecimal:

En la aritmética Hexadecimal(16)(H) ó utilizamos los mismos recursos y a la misma lógica de la aritmética decimal, recurrimos al acarreo de unidades a la posición del exponente siguiente, le sustrae al dígito del exponente inmediato superior, utilizamos la propiedad distributiva y la suma en la multiplicación; y por, en la división usamos el procedimiento de cascada.

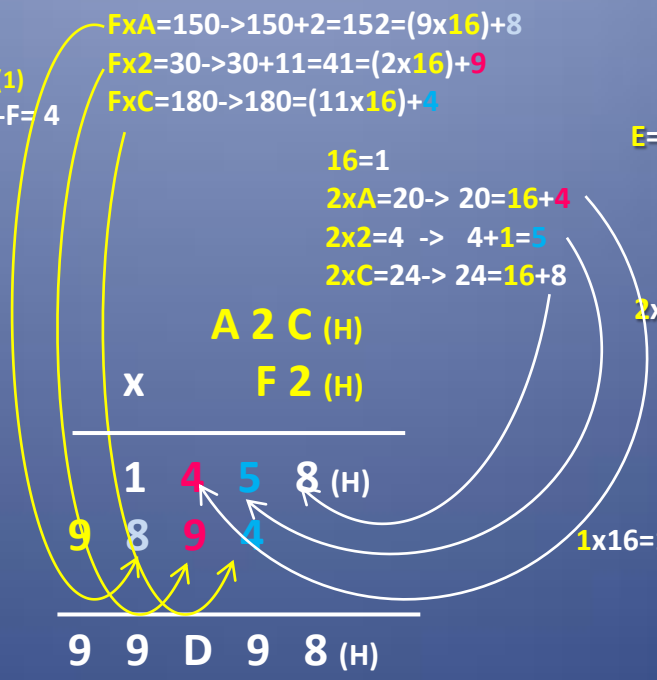
SUMA



RESTA



MULTIPLICACIÓN



DIVISIÓN

VOLVER AL INDICE

3.4. CAMBIO de BASE:

CAMBIO de BASE de un SISTEMA a Otro : Es posible realizar un cambio de base de un sistema numérico a otro teniendo en cuenta su representación polinómica

.

$$110101_{(2)} = 65_{(8)} = 53_{(10)} = 35_{(16)}$$

Notemos que cuanto más grande es la base menor es el numeral representado, manteniendo siempre el mismo valor numérico.

Método 1.

$$3\,C\,F\,6\ (H) = \underbrace{3 \times 16^3}_{} + \underbrace{C \times 16^2}_{} + \underbrace{F \times 16^1}_{} + \underbrace{6 \times 16^0}_{} =$$

$$\overbrace{3 \times 4096}^{} + \overbrace{(C = 12) \times 256}^{} + \overbrace{(F = 15) \times 16}^{} + \overbrace{6 \times 1}^{} =$$

$$\overbrace{12288}^{} + \overbrace{3072}^{} + \overbrace{240}^{} + \overbrace{6}^{} = 1\,5\,6\,0\,6\ (10)$$

Este método principalmente nos permite cambiar a uma base decimal y puede ser usado para cualquier cambio de base no importa lo arbitrario de la base elegida.

3.4. CAMBIO de BASE:

CAMBIO de BASE de un SISTEMA a Otro

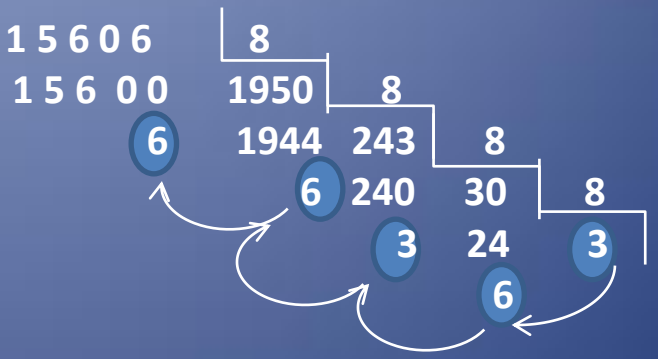
MÉTODO 2:

El cambio de base también puede realizarse usando la conversión a decimal para pasar a cualquier base, de modo que teniendo un numeral decimal, podemos con sucesivas divisiones en forma de cascada resolver el cambio de base deseado.

Método 1.

3 C F 6 (H) = 3 x16^3+C x16^2+F x16^1+6 x16^0 = 1 5 6 0 6 (10)

3 C F 6 (H) = 1 5 6 0 6 (10) = 3 6 3 6 6 (8)



Este método de divisiones sucesivas o de arrastre:

1. El DIVIDENDO es el numeral a cambiar y el DIVISOR la base de cambio,
2. La división NO admite cocientes fracionários,
3. La cascada de divisiones culmina cuando el RESTO ya es indivisible,
4. El nuevo numeral toma como 1er dígito el último COCIENTE , como cifra de mayor exponente,
5. Y construye el numeral em forma “ascendente”, de atrás para delante:. ej.: 3, 6, 3, 6, 6.

3.4. CAMBIO de BASE:

CAMBIO de BASE de un SISTEMA a Otro

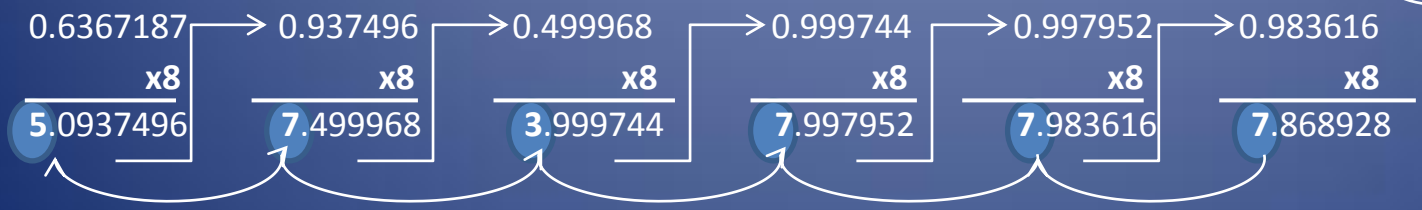
MÉTODO 2 con parte fraccional:

El cambio de base con numerales que contienen parte fraccional también puede realizarse usando la conversión a decimal para pasar a cualquier base, de modo que teniendo un numeral decimal, podemos con sucesivas divisiones de la parte entera, en forma de cascada y multiplicaciones sucesivas, de la parte fraccionaria, resolver el cambio de base deseado.

Método 1.

3 C F 6.A3 (H) = 3 x16³+C x16²+F x16¹+6 x16⁰ +A x16⁻¹ +3 x16⁻² =

3 C F 6.A3 (H) = 1 5 6 0 6.63671875 (10) = 3 6 3 6 6. 777888 (8)



En el caso fraccionario, debemos saber que precisión debe de tener después de la "coma": 2, 4, 6...
Las multiplicaciones deben ser por la base elegida, comenzando desde la parte fraccional conocida y tomando la parte fraccional sucesiva. La parte significativa será la parte entera de la cifra.

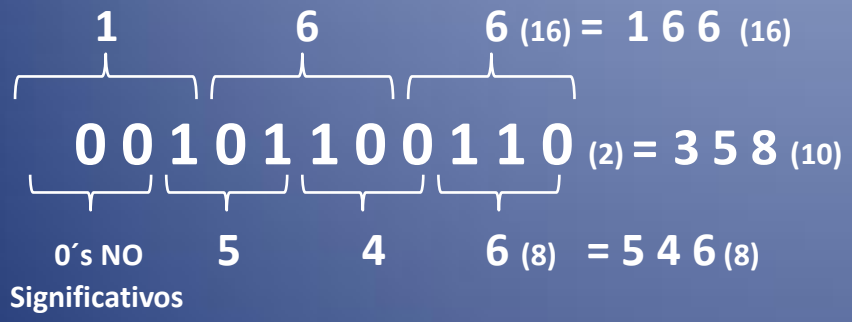
VOLVER AL INDICE

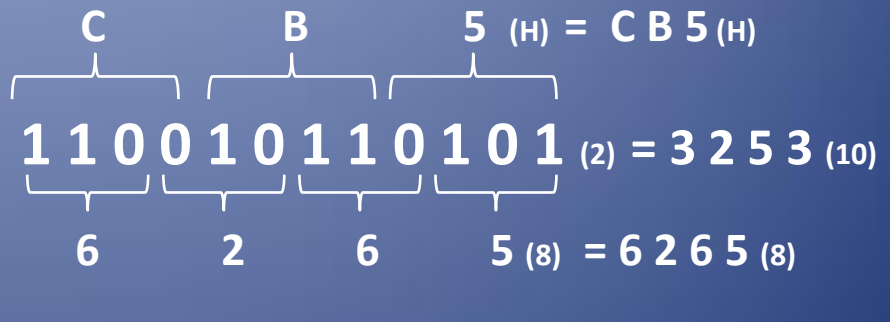
3.4. CAMBIO de BASE:

CAMBIO de BASE de un SISTEMA a Otro :

MÉTODO 3, Otros procesos:

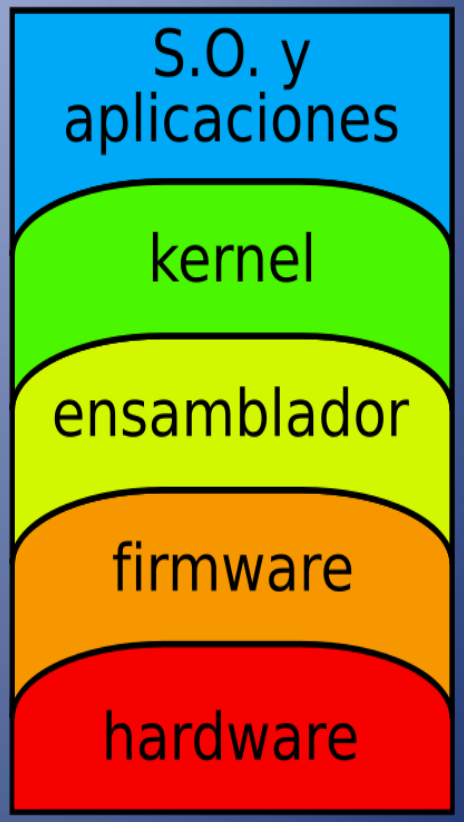
Proceso Empaquetado: Es posible realizar un cambio de base de un sistema numérico binario al Octal o Hexadecimal empaquetando según definición y viceversa.





Con este procedimiento podemos pasar de una base a otra sin usar el procedimiento de transformar a decimal el numeral de base 2, 8 y 16, respetando el empaquetado en bites o Bytes.
En el caso de base Octal se tomarán 3 dígitos correspondientes al desarrollo binario del Byte.
Diferente en el caso Hexadecimal, en el cual tomaremos 4 dígitos que corresponden al desarrollo de dos Bytes.
Viceversa, tomaremos cada cifra ordenadamente y la traduciremos a la base deseada.

4. Representación Interna de Datos:



Todo dato tiene una representación interna en la máquina, el cual a través de un código instruye a la máquina de un proceder específico. Este código o lenguaje entendible por la máquina se lo conoce como "**firmware**", Lenguaje de Máquina o Microcódigo.

El **firmware** es un bloque de instrucciones de programa para propósitos específicos, grabado en una memoria de tipo no volátil (ROM, EEPROM, Flash, etc.), que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo.

Definición del IEEE

El glosario estándar de terminología del software del Institute of Electrical and Electronics Engineers (IEEE), Std 610.12-1990, define el firmware como sigue:

"La combinación de instrucciones de un dispositivo de hardware e instrucciones y datos de computadora que residen como software de solo lectura en ese dispositivo".

4. Representación Interna de Datos:

BIT o bit:

BIT o bit, acrónimo de **B**inary dig**IT**. El bit es la unidad mínima de información empleada en informática, en cualquier dispositivo digital, o en la teoría de la información. Dígito binario refiere a la utilización de dos valores únicos, el 1 y el 0. En el universo máquina, específicamente en electrónica, podemos asignar a cada uno de esos valores el estado de "apagado" (0), y el otro al estado de "encendido" o “abierto” y “cerrado”.

cero 0  **uno 1** 
apagada o encendida
cerrado **abierto**

uno **1**  cero **0**
“ON” “OFF”
cerrado **abierto**

La máquina, sólo tiene la capacidad de entender la apertura o cierre de interruptores eléctricos, el dejar pasar un flujo de electricidad de un valor dado, o no. No comprende otra instrucción.

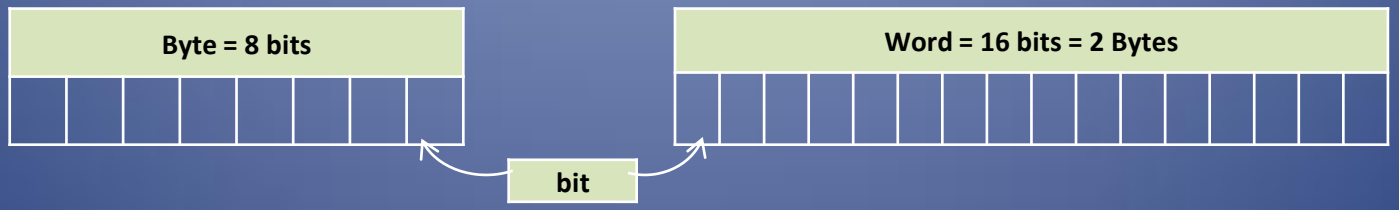
El interprete o compilador será el encargado de traducir el dato impuesto por nosotros en forma de código. A esto se reduce el código binario o bit: es la unidad mínima de información a nivel máquina.

4. Representación Interna de Datos:

Byte u Octeto:

Byte u Octeto: Byte es el acrónimo de **B**inar**Y** **T**upl**E**. también conocido como "byte de 8 bits", reforzando la noción de que era una tupla de *n* bits y que se permitían otros tamaños.
El término "byte" viene de "bite" (en Inglés "mordisco"), como la cantidad más pequeña de datos que un ordenador podía "morder" a la vez.

Byte es el universo compuesto por una secuencia de bits binarios contiguos, cuya cantidad está determinada por elección según el tipo de código (3, 4, 6, 7, 8, 9, 10 y 12).



Un **byte** es la unidad de medida básica para memoria, almacenando el equivalente a un carácter.

El Octeto o Byte de 8 bits, ha llegado a ser casi ubicua. Las variaciones de mayor o menor número de bits se utilizan solo para casos puntuales.

4.1. Representación de Enteros:

Como dijimos anteriormente, la máquina, en este caso la computadora, solo conoce de ceros y unos. Por lo tanto la computadora desconoce la existencia de valores negativos así como de puntos decimales.

Aquí comienza la primer desarrollo de un código primario, establecer la representación de **Enteros = Z**, y Racionales = Q, para poder obtener algebraicamente resultados legibles en el universo computacional.

Así mismo, por las limitaciones propias de la máquina, delimitar los alcances de las representaciones numéricas a los efectos de resultados prácticos. Aquí es donde se aplicará la **Matemática Discreta**, como conjunto acotado de relaciones y sus funciones correspondientes.

En principio, la matemática infinitesimal no puede aplicarse en el computador si no existe una limitante de acción, so pena de caer en resoluciones infinitas e interminables, lo que en computación llamaríamos “entrar en loopings” o “looping”, soluciones no conclusivas.

Se dispondrá, entonces, en la notación binaria, de un primer bit, el más significativo (notación a la izquierda) para determinar el signo del número a representar: NÚMEROS con SIGNO

0 = Positivo

Byte = 8 bits							
0	0	0	0	0	0	0	0

1 = Negativo

Byte = 8 bits							
1	0	0	0	0	0	0	0

4.1. Representación de Enteros:

4.1.1. SIGNO y MAGNITUD (Módulo y Signo):

SIGNO y MAGNITUD, es una de las notaciones con la cual podemos representar los Enteros = $\mathbf{Z}$

La representación binaria SIGNO-MAGNITUD puede representarse genéricamente como:

En el caso de ser positivo **Si** $a_{n-1}=0$ **A=** $\Sigma_{i=0}^{i=n-2} a_i 2^i$; En el caso de ser negativo **Si** $a_{n-1}=1$ **A=** $-\Sigma_{i=0}^{i=n-2} a_i 2^i$

A= valor decimal

VOLVER AL INDICE

4.1. Representación de Enteros:

4.1.1. SIGNO y MAGNITUD (Módulo y Signo):

DESVENTAJAS de la representación **Signo y Magnitud**:

1. Las operaciones aritméticas de SUMA y RESTA requieren consideraciones especiales, en cuanto a los signos de los números como en sus magnitudes relativas, a fin de poder realizar dichas operaciones.
2. Existen dos (2) representaciones del CERO (0): +0 = **0** 0000000
   - 0 = 1 0000000
3. La cantidad de bit requerida para la representación numérica, varía el valor de la misma:
   Ej.: 115(10) = **0** 1 1 1 0 0 1 1 requiere de 8 bits
   -218 (10) = **1** 1 1 0 1 1 0 1 0 requiere de 9 bits
   -305 (10) = **1** 1 0 0 1 1 0 0 0 1 requiere de 10 bits
   No se logra con ello una estandarización.

VENTAJAS de la representación **Signo y Magnitud :**

1. Posee un rango simétrico: los números van del $+127_{10}$ = **0**1111111_2
2. pasando por el $+0_{10}$ = **0**0000000_2 y el -0_{10} = **1**0000000_2
3. hasta el -127_{10} = **1**1111111_2.

VOLVER AL INDICE

4.1. Representación de Enteros:

4.1.2. COMPLEMENTO a 1:

COMPLEMENTO a 1, es una de las notaciones con la cual podemos representar los Enteros = $\mathbb{Z}$

El formato de Complemento a uno que nos permite codificar en binario en punto fijo con 8 bits (un byte), al igual que con la representación en Signo y Magnitud, esto nos otorga 1 bit para el signo y 7 bits para la magnitud.
Con 8 bits, podemos representar, en teoría al menos $2^8 = 256$ números. Los cuales, según éste formato, van a estar repartidos entre 128 números positivos (bit de signo en 0) y 128 números negativos (bit de signo en 1).
Un número Entero negativo se representará colocando como bit de signo un 1 y como mantisa su complementario en valor absoluto.

Ej.: $-97_{10} \longrightarrow |-97_{10}| = \overline{97}_{10}$

$97_{10} = 1100001_2 \longrightarrow \overline{97}_{10} = 0011110_2$

$-97_{10} = \overline{97}_{10}$ (complementario de 97_{10})

1 por Negativo

SIGNO

C1 (1100001) = 0011110

MAGNITUD

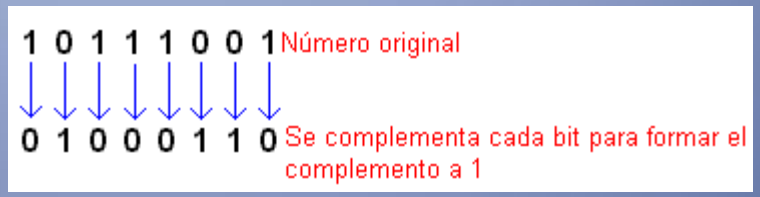
Complementario a 1							
1	0	0	1	1	1	1	0

Al representar en Complemento a uno, aparece nuevamente el cero signado: **0**0000000$_2$ (+0$_{10}$) y **1**1111111$_2$ (-0$_{10}$).

VOLVER AL INDICE

4.1. Representación de Enteros:

4.1.2. COMPLEMENTO a 1:



DESVENTAJAS de la representación **COMPLEMENTO a 1** :

1. Posee doble representación del cero. Al representar en Complemento a uno, aparece nuevamente el cero signado: **+0** = **0**0000000$_2$ (+0$_{10}$) y
   - 0 = 11111111$_2$ (-0$_{10}$)

VENTAJAS de la representación **COMPLEMENTO a 1 :**

1. Posee un rango simétrico: los números van del +127$_{10}$ = **0**1111111$_2$, pasando por el +0$_{10}$ = **0**0000000$_2$ y el -0$_{10}$ = **1**1111111$_2$, hasta el -127$_{10}$ = **1**0000000$_2$. Y en forma general, para *n*-bits, el rango (en decimal) para Complemento a uno es ($-2^{n-1}-1$; $2^{n-1}-1$), o bien $\pm 2^{n-1}-1$.

2. Permite operar aritméticamente. **NOTA**: al operar se debe sumar el acarreo obtenido al final de la adición/resta realizadas (conocido como **end-around carry**), en caso de haberlo obtenido, para conseguir el resultado correcto.

4.1. Representación de Enteros:

4.1.2. Aritmética de COMPLEMENTO a 1:

Por ejemplo: $21_{10} + (-97_{10}) = -76_{10}$

1er Paso: Resolver $-97_{10} = |\overline{-97_{10}}| = \overline{97_{10}} = |\overline{1100001_2}| = \mathbf{1}0011110_2$

2do Paso: Resolver $21_{10} + \overline{97_{10}} = 10101_2 + 011110_2 = 0110011_2$

$$\begin{array}{rr} 10101_2 & 21_{10} \\ + & + \\ \mathbf{1}0011110_2 & (-97_{10}) \\ \hline 10110011_2 & (-76_{10}) \end{array}$$

Por ejemplo: $2_{10} + (-1_{10}) = 1_{10}$

1er Paso: Resolver $-1_{10} = |\overline{-1_{10}}| = \overline{1_{10}} = |\overline{00000001_2}| = \mathbf{1}1111110_2$

2do Paso: Resolver $2_{10} + \overline{1_{10}} = 00000010_2 + \mathbf{1}1111110_2 = 0110011_2$

$$\begin{array}{rr} 00000010_2 & 2_{10} \\ + & + \\ \mathbf{1}1111110_2 & (-1_{10}) \\ \hline \mathbf{1}00000000_2 & \downarrow \\ \text{(end-around carry) acarreo circular} \rightarrow \quad \mathbf{1} & \\ \hline 00000001_2 & 1_{10} \end{array}$$

Por ejemplo: $1001_2 + (-1111_2)$

1er Paso: Resolver $-1111_2 = |-1111_2| = \mathbf{1}0000_2$

2do Paso: Resolver $1001_2 + (\overline{-1111_2}) = 1001_2 + \mathbf{1}0000_2 = \overline{110011_2} = (-110_2)$

El acarreo final circular es 0 y por tanto

VOLVER AL INDICE

4.1. Representación de Enteros:

4.1.2. Corolario de COMPLEMENTO a 1:

Los protocolos de internet IPv4, ICMP, UDP y TCP usan todos el mismo algoritmo de suma de verificación de 16 bits en complemento a uno.

Aunque la mayoría de la computadoras carecen del hardware para manejar acarreo del último bit (end-around carry), la complejidad adicional es aceptada ya que es igualmente sensible a errores en todas las posiciones de bits.

En UDP, una representación de todos ceros indica que la suma de verificación opcional ha sido omitida.

La otra representación, todos unos, indica un valor 0 en la suma de verificación (las sumas de verificación son obligatorias para IPv4, TCP e ICMP; fueron omitidas en IPv6).

4.1.3 COMPLEMENTO a 2:

COMPLEMENTO a 2 Es necesario **solamente** cuando vamos a buscar la representación de un número negativo o cuando vamos a leer un formato que corresponde a negativo. Es un formato para representar números con signo fundamentado en el sistema posicional de escritura con base dos: el sistema binario. Primero, se establece el espacio de trabajo, es decir, el número de posiciones binarias de escritura o **bits**. A **m** bits tenemos 2^m representaciones disponibles. Con 8 bits, podemos representar, $2^8 = 256$ números, los cuales, según éste formato, van a estar repartidos entre 128 números positivos (bit de signo en 0) y 128 números negativos (bit de signo en 1). La representación del cero y la asignada a cada entero positivo corresponde a su escritura en sistema binario, tan sólo añadimos ceros al frente para completar el total de bits a la escritura fija de ser necesario. Reconoceremos que una representación corresponde a un número positivo porque siempre comienza con un bit de Cero. Un número Entero negativo se representará colocando como bit de signo un **1** y como mantisa su complementario en valor absoluto y todos sus ceros menos significativos los invertiremos a **unos** .

Ej.: $-97_{10} \longrightarrow |-97_{10}| = \overline{97}_{10}$

$97_{10} = 1100001_2 \longrightarrow \overline{97}_{10} = 0011110_2$

$-97_{10} = \overline{97}_{10}$ (complementario de 97_{10})

1 por Negativo

SIGNO

C1 (1100001) = 0011110

MANTISA

Complementario a 1							
1	1	1	1	1	1	1	0

4.1.3 COMPLEMENTO a 2:

COMPLEMENTO a 2: Su utilidad principal se encuentra en la sustracción, dado que el procesador solo conoce la adición (suma de binarios⟶ binarios negativos). La resta de un número binarios positivo (minuendo) y otro negativo (sustraendo), puede obtenerse **sumando al minuendo el complemento a dos (C 2) del sustraendo**. Los números binarios positivos se mantienen inalterables

El complemento a 2 de un numero binario se obtiene tomando el complemento a 1, y sumándole 1 al bit menos significativo. A continuación se ilustra este proceso para el numero 1001 = 9

```
9 = 1001
    0110  ⟶ Complemento a 1
+      1  ⟶ Se suma 1 al LSB
  ------
    0111  ⟶ Complemento a 2
```

⟶ Representación de **-9** en un Byte de 8 bits

Complementario a 1en 8 bits							
1	1	1	1	0	1	1	1

Este es un sistema que nos permite representar números binarios de forma negativa, en dónde el bit más significativo (MSB: most significant bit) es el bit del signo.
Cuando se agrega el bit del signo 1 al MSB en un Byte de 8 bits, el número Complemento a 2 con signo se convierte en 11110111 y es el número equivalente a – 9. Los Ceros menos significantes se invierten hasta completar la palabra.

4.1. Representación de Enteros:

4.1.2. Aritmética de COMPLEMENTO a 2:

El Complemento a 2 de un número binario se puede considerar equivalente a su negativo de tal manera que la RESTA de un numeral positivo y otro negativo puede resolverse a través de la suma binaria:

$$n_{(a-b)} = a + (C2 \text{ de } b)$$

En caso de restar sumando utilizando el complemento a 1 (C1), el 1 de arrastre que sale fuera del registro se suma nuevamente al resultado, quedando en el registro el resultado de la resta.

Por ejemplo: $2_{10} + (-1_{10}) = 1_{10}$

1er Paso: Resolver $-1_{10} = |-1_{10}| = 1_{10} = |00000001_2| = \mathbf{1}1111110_2$

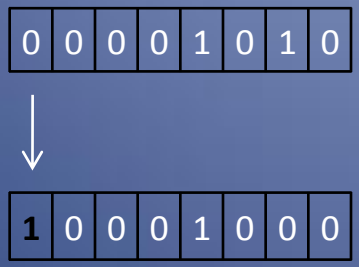
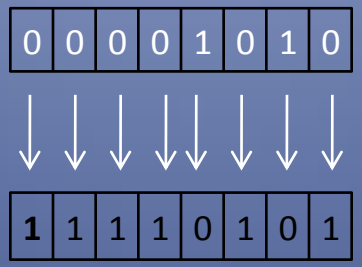
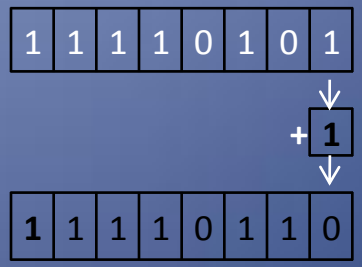
2do Paso: Resolver $2_{10} + 1_{10} = 00000010_2 + \mathbf{1}1111110_2 = 0110011_2$

$$\begin{array}{ll} 00000010_2 & 2_{10} \\ + & + \\ \mathbf{1}1111110_2 & (-1_{10}) \\ \hline \mathbf{1}00000000_2 & \downarrow \\ \text{(end-around carry) acarreo circular} \curvearrowright 1 & \\ \hline 00000001_2 & 1_{10} \end{array}$$

VOLVER AL INDICE

4.1. Representación de Enteros:

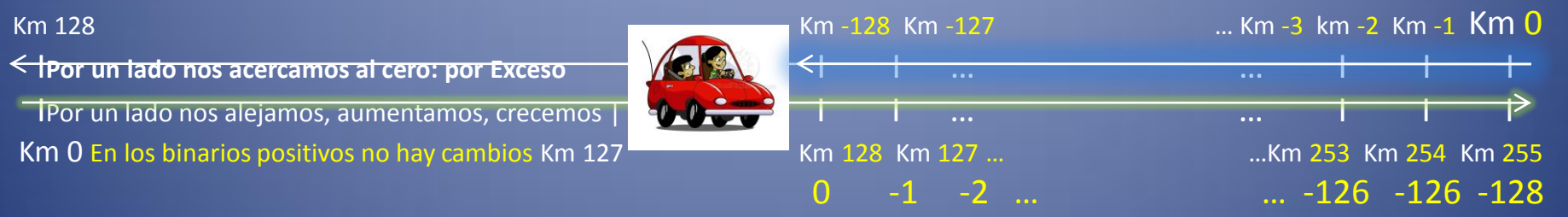
4.1.2. Resumen Simplificado:

SIGNO y MAGNITUD	COMPLEMENTO a 1	COMPLEMENTO a 2
Para diferenciar el positivo del negativo solo se le cambia el bit del signo (0 x 1)	Para construir el negativo basta con invertir los ceros y unos y por ende, cambiar el bit del signo	Para convertir en Complemento a 2, al Complemento a 1 se le debe adicionar (sumar) 1
binario Normal	binario Normal	Complemento a 1
		
Signo y Magnitud	Complemento a 1	Complemento a 2

4.1.4 EXCESO a 2^{n-1} :

EXCESO a 2^{n-1}: Por lo general se considera positivos a los 128 primeros (desde 0 a 127) y negativos a los 128 restantes (desde 128 a 255). Aunque en el Z-80 los considera casi siempre, todos positivos

De esta forma, 255 sería equivalente a "-1", 254 a "-2", 253 a "-3", y así sucesivamente hasta 128 que sería en realidad, "-128".



De Forma paralela con el recorrido de un automóvil, que mientras avanza aumenta sus Km y a su vez disminuye la distancia al destino, lo mismo sucede con el Exceso a 2^{n-1} disminuye de manera proporcional respecto al Complemento a 2.

Tabla Representación de Enteros=Z en 8 bits (Simplificada)

	Decimal	Complemento a 2	EXCESO a 2^n-1		Decimal	Complemento a 2	EXCESO a 2^n-1		Decimal	Complemento a 2	EXCESO a 2^n-1
POSITIVOS	127	01111111	00000000	POSIT	002	00000010	11111101	NEGATIVOS	...	...	...
	126	01111110	00000001		001	00000001	11111110		125	10000011	00000011
	125	01111101	00000010		000	00000000	11111111		126	10000010	00000010
	124	01111100	00000011	NEG	001	11111111	01111110		127	10000001	00000001
	...	...	...		002	11111110	01111101		128	10000000	00000000

VOLVER AL INDICE

4.1.4 EXCESO a 2^{n-1} :

EXCESO a 2^{n-1}: Este procedimiento no utiliza bit de signo y considera toda cifra complementaria a 2 como positiva, estableciendo como resultante lea diferencia entre ésta y el rango mayor. En el caso de una cifra de 8 bits será la diferencia entre el C 2 y 127 para números menores a Cero; en el caso de numerales mayores o iguales a Cero, la diferencia se establecerá contra 128, debido a que los rangos de C2 no son simétricos. A ésta diferencia ente C2 y Exceso a 2 se la denomina "desplazamiento".

Pongamos un ejemplo (byte=8 bits):

DECIMAL:	BINARIO	EXCESO **a 2^{n-1}** :
118	00111010	00001001
-118	10001111	00001001
-125	10000100	00000010

Para un valor en base 10 de un número entero (**N**) escrito en Exceso a 2^{n-1}, se utiliza la fórmula:

$$N_{ex} = \left[\left(\sum_{i=0}^{i=n-1} a_i * 2^i \right) - 2^{n-1} \right]_{10}$$

$$00001001_{ex\,a\,-118} = \left[\left(\sum_{i=0}^{i=n-1} a_i * 2^i \right) - 2^{n-1} \right]_{10} = \left[\left(1 \cdot 2^3 + 1 \cdot 2^0 \right) - 2^7 \right]_{10} = \left[(8 + 1) - 127 \right]_{10} = -118_{10}$$

4.1.4 EXCESO a 2^{n-1} :

EXCESO a 2^{n-1}: Este procedimiento no utiliza bit de signo y considera toda cifra complementaria a 2 como positiva, estableciendo como resultante lea diferencia entre ésta y el rango mayor. En el caso de una cifra de 8 bits será la diferencia entre el C 2 y 127 para números menores a Cero; en el caso de numerales mayores o iguales a Cero, la diferencia se establecerá contra 128, debido a que los rangos de C2 no son simétricos. A ésta diferencia ente C2 y Exceso a 2 se la denomina "desplazamiento".

Pongamos un ejemplo (byte=8 bits):

DECIMAL:	BINARIO	EXCESO **a 2^{n-1}** :
118	00111010	00001001
-118	10001111	00001001
-125	10000100	00000010

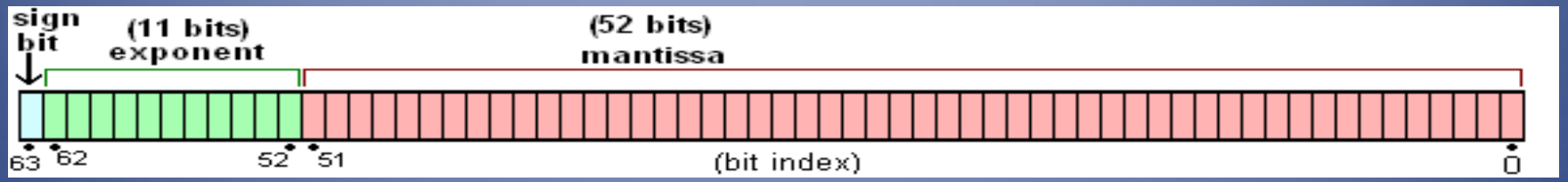
Para un valor en base 10 de un número entero (**N**) escrito en Exceso a 2^{n-1}, se utiliza la fórmula:

$$N_{ex} = \left[\left(\sum_{i=0}^{i=n-1} a_i * 2^i \right) - 2^{n-1} \right]_{10}$$

$$00001001_{ex\,a\,-118} = \left[\left(\sum_{i=0}^{i=n-1} a_i * 2^i \right) - 2^{n-1} \right]_{10} = \left[\left(1 \cdot 2^3 + 1 \cdot 2^0 \right) - 2^7 \right]_{10} = \left[(8 + 1) - 127 \right]_{10} = -118_{10}$$

4.2 Reales, Punto FLOTANTE

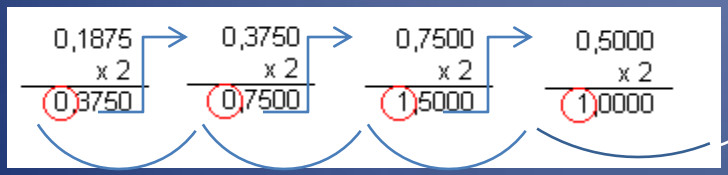
La representación de los números Reales con coma flotante en los sistemas de numeración del computador, están basados en la notación científica exponencial, de modo que su formato pueda ser representado en un espacio normativo llamado **Norma IEEE 754**, de 32 ó 64 bits, los cuales están divididos en tres (3) bloques. Cada bloque representa una parte del número representado. De izquierda a derecha, el primer bloque tiene una capacidad de 1 bit y se utiliza para representar el signo (0 = positivos y 1= negativos). El segundo bloque con capacidad de 8 ó 11 bits para representar el exponente y el bloque de más a la derecha, de 23 ó 52 bits para la representación de la mantisa o número representado.
Existe también una representación llamada EXTENDIDA, que contiene 80 bits ó 10 Bytes.

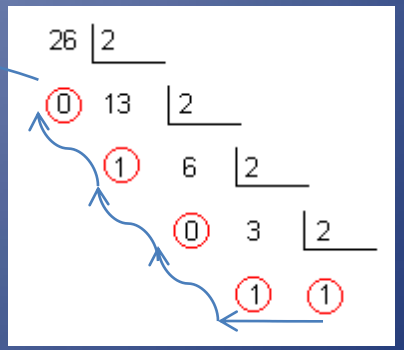


$26{,}1875_{10}$ → $26_{10} = 1101_2$

$26{,}1875_{10} = 11010{,}0011_2$

$0{,}1875_{10} = 0{,}0011_2$





$26{,}1875_{10} = 2{,}61875 \times 10^{1} = 11010.0011 = 1.10100011 \times 2^{4} = 1.10100011 \times 2^{0100}$

OVERFLOW MANTISA BASE EXPONENTE

Signo
0=positivo
1=Negativo

CEROS NO SIGNIFICATIVOS

EXPONENTE

MANTISA

CEROS NO SIGNIFICATIVOS

(NORMA IEEE 754 de 32 bits)

4.2.1 Aritmética con Punto Flotante

1 bit	8 bits	23 bits
Signo del número	Exponente sesgado	Mantisa. Parte fraccionaria f

En el caso de *simple precisión*, o de 32 bits, el valor del **(e)** varía de 0 = $(00000000)_2$ hasta 255 = $(11111111)_2$ para enteros positivos y no permitiría representar negativos; por esa razón el almacenamiento de dicho valor se realiza en forma sesgada como un valor NO negativo **(s)**. Para determinar dicho valor de almacenamiento o valor sesgado del exponente **(v)**, se toma de forma arbitraria un cero o punto medio al *centro* de este rango de valores, obteniendo una partición a la izquierda de 127 bits y una partición a la derecha de 128 bits. De esta manera podremos registrar los valores positivos y negativos del exponente **(e)** en forma positiva. Para determinar dichos valores utilizaremos la siguiente fórmula:

$$e = v - s$$

Para poder realizar esta conversión de forma manual directamente, podemos utilizar en el cálculo primario la numeración decimal, traduciéndola a binaria para poderla registrar en su representación mecanicista, o de modo inverso, podemos desde la representación mecanicista re-expresarla en forma decimal para su cálculo.

En el caso anterior, donde **$26{,}1875_{10}$** es equivalente a **$1.10100011 \times 2^{4} = 1.10100011 \times 2^{0100}$**,
el exponente de la base es 4 (**e = 4**),
entonces ***e = v – s*** es **4 = v – 127**,
de lo que deducimos que **v = 131** o lo que es igual a **v = 10000011**

VOLVER AL INDICE

4.2.1 Aritmética con Punto Flotante

Otro ejemplo podría ser:

Sea X = 13,625 = 1101.101_2

$1101.101_2 = 1.\mathbf{101101}_2 \times 2^{\mathbf{3}}$

Entonces si ***e = v – s*** , y conocemos **e=3 = v – 127**,

deducimos que **v = 130** o lo que es igual a **v = 10000010**

SIGNO del NÚMERO	EXPONENTE SESGADO	PARTE FRACCIONARIA de la MANTISA
0	1000001	10110100000000000000000

Debemos recordar que la notación en el bloque de la mantisa es solamente de la parte fraccionaria del numeral, quedando fuera el primer 1 (uno), o parte entera, debido a que por norma siempre está presente y es una forma de aprovechamiento del recurso espacial de la memoria. Los ceros de relleno completan los espacios libres y carecen de valor.

4.2.1 Aritmética con Punto Flotante

e_{10} = exponente en base decimal;

V_{10} = valor sesgado en base decimal;

$e_{10}(-e_{10})$ =

K_S = constante de sesgamiento ;

V_2 = Exponente Sesgado en **base binaria**

Tabla de valores del EXPONENTE SESGADO				
e_{10}	$V_{10} = e_{10}(-e_{10}) + K_S$	S_{10}	V_2 = Exponente Sesgado	
e = -127	V = 127 – 127	0	0000	0000
e = -126	V = 127 – 126	1	0000	0001
...	...	...	...	...
e = -15	V = 127 – 15	112	0111	0000
e = -14	V = 127 – 14	113	0111	0001
e = -13	V = 127 – 13	114	0111	0010
e = -12	V = 127 – 12	115	0111	0011
e = -11	V = 127 – 11	116	0111	0100
e = -10	V = 127 – 10	117	0111	0101
e = -9	V = 127 – 9	118	0111	0110
e = -8	V = 127 – 8	119	0111	0111
e = -7	V = 127 – 7	120	0111	1000
e = -6	V = 127 – 6	121	0111	1001
e = -5	V = 127 – 5	122	0111	1010
e = -4	V = 127 – 4	123	0111	1011
e = -3	V = 127 - 3	124	0111	1100
e = -2	V = 127 - 2	125	0111	1101
e = -1	V = 127 - 1	126	0111	1110
e = 0	V = 0 + 127= 127	127	0111	1111
e = 1	V = 1 + 127	128	1000	0000
e = 2	V = 2 + 127	129	1000	0001
e = 3	V = 3 + 127	130	1000	0010
e = 4	V = 4 + 127	131	1000	0011
e = 5	V = 5 + 127	132	1000	0100
e = 6	V = 6 + 127	133	1000	0101
e = 7	V = 7 + 127	134	1000	0110
e = 8	V = 8 + 127	135	1000	0111
e = 9	V = 9 + 127	136	1000	1000
e = 10	V = 10 + 127	137	1000	1001
e = 11	V = 11 + 127	138	1000	1010
e = 12	V = 12 + 127	139	1000	1011
e = 13	V = 13 + 127	140	1000	1100
e = 14	V = 14 + 127	141	1000	1101
...	...	...	...	...
e = 127	V = 127 + 127	254	1111	1110
e = 128	V = 128 + 127	255	1111	1111

VOLVER AL INDICE

MATERIAL REFERENCIAL

M Morris Mano -	Lógica Digital y Diseño De Computadores	
J. L. Hennessy-D. Patterson	Arquitectura de Computadores	Mc Graw-Hill
William Stallings	Org. y Arquitectura de computadores	Prentice Hall
Andrew Tanenbaum	Organización de Computadoras	Prentice Hall
Erika Vilches	Organización Computacional	ITSM 2008
Ing. J.L. Jiménez	Teoría Electrónica	Ladelec
Hans Rautenberg	Sistemas numéricos, Diseño de Circ.	Univ. Concepción, CL
Carlos J. Pes Rivas		

Propósito General:

El propósito de realizar ésta presentación, es el abordar los temas del bolillado de Organización del Computador I, como guía e introducción a los diferentes temas. La profundización y ampliación de los mismos la puede obtener a través de la bibliografía citada.

Todos los aportes posibles, serán recibidos y analizados para su posible incorporación en la reedición de este material.

Correo de recepción de aportes: adolfomontielvalentini@hotmail.com

Análisis y compilación del material ante expuesto:

Adolfo Montiel Valentini ® 2011

Prof. Daniel Zavadski– Organización del Computador I
Instituto Normal de Enseñanza Técnica (INET)

VOLVER AL INDICE