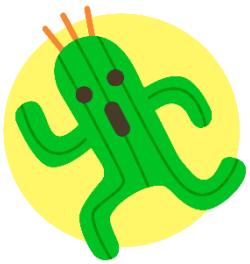
Fundamentos de Computadores

(1) Representación de la información



V 0.2 2024_09_28

Aprende sin espinas
con @carlos_cactus

Sócrates se equivocaba. El conocimiento no es lo único que crece al compartirse: La alegría también.

A la inspiración del bucle_infinito,
al Cibergrupo y al tHash_A, por su amistad,
y sobre todo, a quienes dicen "pero quiero"
cuando sienten "no puedo".

¡Un saludo sin espinas!
@carlos_cactus :D

Y si quieres saber más:
¡Encuéntrame en Telegram como @carlos_cactus o habla con Espinito, el bot Sin Espinas, en @GestionSinEspinasBot.

Únete a la comunidad de Telegram Sin Espinas y no te pierdas nada!

Deja de preocuparte por aprobar y ¡Aprende sin Espinas!

1. Sistemas de representación numéricos

Un sistema de numeración es un método para la representación de conjuntos de valores numéricos.

El número en sí es un concepto abstracto, y aunque se corresponde con su representación, NO ES SU REPRESENTACIÓN.

Para la definición de operaciones y relaciones entre los números, se aprovecha que son conjuntos ordenados: tienen un ORDEN DE PRECEDENCIA.

Se destacan varios tipos de sistemas de representación:

- Sistemas de raíz o base
- Sistema de dígitos firmados
- Sistemas de residuos
- Sistemas racionales

Los sistemas de numeración basados en raíz o base describen los valores de los números en función de un valor raíz, la base del sistema. La base coincide con el número de dígitos disponibles para la representación.

Un dígito es un signo gráfico para representar números.

En base 10, se dispone de 10 dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
En base 2, se dispone de 2 dígitos: 0, 1
En base 16, se tienen 16: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

La base en que se expresa una cantidad e denota como subíndice. Se dice que $203_{(a}$ es “203 en base a”, que no tiene por qué corresponder con $203_{(c}$.

Si un sistema de raíz utiliza una sola base, es denomina sistema de base fija (contrapuesto a los sistemas de base mixta, ejemplo de los cuales es el sistema horario de un reloj).

Un sistema de numeración posicional es todo aquel en que el valor del dígito se determina mediante el orden secuencial en que se encuentran.

En este manual, se tratan sistemas posicionales de base fija.

2. Sistema de numeración posicionales

Se denomina PESO a la posición que ocupa un dígito en una secuencia ordenada que representa un número y que define el valor de ese dígito en la secuencia.

EJEMPLO 1 :

$$302{,}56 = 3 \cdot 10^2 + 2 \cdot 10^0 + 5 \cdot 10^{-1} + 6 \cdot 10^{-2}$$

Nótese que:

$$10^{-n} = \frac{1}{10^{+n}}$$

Por tanto, el peso de un dígito está determinado por su posición en la secuencia y por la base del sistema.

El PESO de un dígito en la posición p para una base de numeración b es:

$$PESO_p = p^b$$

Para todo dígito x_i en una secuencia, de acuerdo con su posición i, se define:

$$x_{n-1}x_{n-2}x_{n-3} \dots x_1x_0x_{-1}x_{-2} \dots x_{-m}$$

Nótese que

a) Para una base b, se tiene $0 \leq x_i \leq b-1$. Es decir, hay tantos dígitos distintos como b indica, cuyos valores van de 0 a b – 1. Por ejemplo, en base 3, se tendrían 3 dígitos 0, 1 y 2 y van de 0 a 3 – 1 = 2.
b) Las posiciones de x_i con subíndice negativo denotan la parte FRACCIONARIA del número, que se separa de la parte entera o no fraccionaria por una COMA, de modo que las posiciones con subíndice positivo corresponden a la parte entera.
c) Aquí se DEBE EVITAR EL TERMINO “PARTE DECIMAL” para designar a la parte FRACCIONARIA para evitar ambigüedades y confusiones en cuanto a la base.
d) Se entiende por significación de un bit la medida en que influye en la determinación de la magnitud representada:
    - El bit más significativo es el de mayor peso (más a la izquierda en la secuencia).
    - El bit menos significativo es el de menor peso (más a la derecha en la secuencia).

El sistema en base 10 se denomina sistema DECIMAL.
El sistema en base 16 se denomina sistema HEXADECIMAL.
El sistema en base 8 se denomina sistema OCTAL.
El sistema en base 2 se denomina sistema BINARIO

De esto se desprende el TEOREMA FUNDAMENTAL DE LA NUMERACIÓN (TFN):

$$X = \sum_{i=-m}^{n-1} x_i \cdot b^i = x_{n-1} \cdot b^{n-1} + x_{n-2} \cdot b^{n-2} + \cdots + x_{-m} \cdot b^{-m}$$

Esto conlleva que para determinar el valor de una secuencia de dígitos hace falta conocer el peso de cada dígito que la forma y la base en que se expresa la cantidad.

3. Cambio de base

Los métodos de cambio de base permiten calcula la secuencia de dígitos de un valor expresado en una base original b_o en una nueva base de destino b_d, es decir:

$$cambio\ de\ base\ (x_{(b_o}) = x_{(b_d}$$

Se consideran 2 técnicas, cuya combinación permite el tránsito entre cualquier par de bases de origen y destino:

- Mediante TFN
- Mediante el TEOREMA DEL RESTO

3.1. Aplicación de TFN (de base b ≠ 10 hacia base 10)

1) Primero se expresan los dígitos en la base de origen de acuerdo con el TFN.

2) Se convierten todos los dígitos de la base de origen mayores que 10 a base 10 de acuerdo con: Para la base hexadecimal, se tiene:

$A_{(16} = 10_{(10}$ $B_{(16} = 11_{(10}$ $C_{(16} = 12_{(10}$
$D_{(16} = 13_{(10}$ $E_{(16} = 14_{(10}$ $F_{(16} = 15_{(10}$

3) Se opera la expresión resultante en la base de destino.

EJEMPLO 2 : CAMBIO DE BASE MEDIANTE TFN

Se desea expresar $321_{(4}$ en base 10:

Basta con operar en base 10 considerando el peso que tiene cada bit en la base de origen:

$$321_{(4} = 3 \cdot 4^2 + 2 \cdot 4^1 + 1 \cdot 4^0 = 57_{(10}$$

3.2. Aplicando el TEOREMA DEL RESTO (de base 10 hacia base b ≠ 10):

Se usa de forma distinta según si se desea conocer la parte ENTERA o la parte FRACCIONARIA, denominada MANTISA (dígitos a la derecha de la coma).

Para la parte ENTERA:

- Hacer sucesivas DIVISIONES de los dígitos ENTEROS de la base de salida usando como DIVISOR la NUEVA BASE.
- Se opera en base 10.
- Los RESIDUOS que aparecen se toman como DÍGITOS EN LA NUEVA BASE.
- Los residuos se toman en SENTIDO CONTRARIO al de aparición (el más RECIENTE ocupa la posición de MÁS PESO en la secuencia obtenida: "de abajo hacia arriba").

Para definir la MANTISA:

- Se MULTIPLICA la MANTISA (parte fraccionaria) por la NUEVA BASE.
- Se separa el resultado del producto en una nueva mantisa y una parte entera.
- Se opera en base 10 hasta que se obtiene un COCIENTE NULO.
- Se sigue multiplicando hasta que se observe alguna de las siguientes 2 situaciones:
  - Encontrar un comportamiento periódico.
  - Alcanzar el número de dígitos admitidos por la representación deseada.
- Se toman los dígitos en el MISMO SENTIDO con que se calcularon (el más RECIENTE ocupa la posición de MENOS PESO en la secuencia obtenida: "de arriba hacia abajo").

La secuencia en la nueva base es la suma de la parte entera y la mantisa calculadas: basta con unirlas.

EJEMPLO 3 : CAMBIO DE BASE MEDIANTE TEOREMA DEL RESTO

Se desea expresar $57_{(10}$ en base 3.

Se descomponen sucesivamente los dígitos de salida:

$$\underbrace{57}_{\substack{secuencia\\ en\ base_o}} = \underbrace{3}_{\substack{nueva\\ base}} \cdot \underbrace{19}_{cociente} + \underbrace{0}_{resto}$$

$$\underbrace{19}_{\substack{cociente\ previo\\ en\ base_o}} = \underbrace{3}_{\substack{nueva\\ base}} \cdot \underbrace{6}_{cociente} + \underbrace{1}_{resto}$$

$$\underbrace{6}_{\substack{cociente\ previo\\ en\ base_o}} = \underbrace{3}_{\substack{nueva\\ base}} \cdot \underbrace{2}_{cociente} + \underbrace{0}_{resto}$$

$$\underbrace{2}_{\substack{cociente\ previo\\ en\ base_o}} = \underbrace{3}_{\substack{nueva\\ base}} \cdot \underbrace{0}_{cociente} + \underbrace{2}_{resto}$$

Ahora se ORDENAN LOS RESTOS en SENTIDO ASCENDENTE, en orden contrario a su aparición (primero el más reciente):

$$57_{(10} = 2010_{(3}$$

EJEMPLO 4 : CAMBIO DE BASE CON MATINSA (PARTE FRACCIONARIA)

Se desea expresar $21{,}43_{(10}$ en base 2, truncando a 8 bits fraccionarios.

Para la parte entera:

Como se desea una base distinta de 10, se usa el teorema del resto:

$$\underbrace{21}_{\substack{dígitos\\ en\ base_o}} = \underbrace{2}_{\substack{nueva\\ base}} \cdot \underbrace{10}_{cociente} + \underbrace{1}_{resto}$$

$$\underbrace{10}_{\substack{cociente\\ previo}} = \underbrace{2}_{\substack{nueva\\ base}} \cdot \underbrace{5}_{cociente} + \underbrace{0}_{resto}$$

$$5 = 2 \cdot 2 + 1$$

$$2 = 2 \cdot 1 + 0$$

$$\underbrace{1}_{\substack{cociente\\ previo}} = \underbrace{2}_{\substack{nueva\\ base}} \cdot \underbrace{0}_{cociente} + \underbrace{1}_{resto}$$

Se detiene cuando se obtiene un cociente nulo. Es decir:

$$21_{(10} = 10101_{(2}$$

Para la parte fraccionaria:

Se aplica el teorema del resto MULTIPLICANDO por la nueva base.

Para cada producto obtenido, se separa parte entera de mantisa:

- La mantisa resultante se usa para iterar.
- La parte entera se toma como dígito en la nueva base

$$\underbrace{0{,}43}_{\substack{mantisa\\ original}} \cdot \underbrace{2}_{\substack{nueva\\ base}} = 0{,}86 = \underbrace{0{,}86}_{\substack{nueva\\ mantisa}} + \underbrace{0}_{\substack{parte\ entera:\\ DÍGITO\ EN\ b_d}}$$

$$\underbrace{0{,}86}_{\substack{mantisa\\ obtenida}} \cdot \underbrace{2}_{\substack{nueva\\ base}} = 1{,}72 = \underbrace{0{,}72}_{\substack{nueva\\ mantisa}} + \underbrace{1}_{\substack{parte\ entera:\\ DÍGITO\ EN\ b_d}}$$

$$\underbrace{0{,}72}_{\substack{mantisa\\ obtenida}} \cdot \underbrace{2}_{\substack{nueva\\ base}} = 1{,}44 = \underbrace{0{,}44}_{\substack{nueva\\ mantisa}} + \underbrace{1}_{\substack{parte\ entera:\\ DÍGITO\ EN\ b_d}}$$

$$0{,}44 \cdot 2 = 0{,}88 = 0{,}88 + 0$$

$$0{,}88 \cdot 2 = 1{,}76 = 0{,}76 + 1$$

$$0{,}76 \cdot 2 = 1{,}52 = 0{,}52 + 1$$

$$0{,}52 \cdot 2 = 1{,}04 = 0{,}04 + 1$$

$$\underbrace{0{,}04}_{\substack{mantisa\\ obtenida}} \cdot \underbrace{2}_{\substack{nueva\\ base}} = 0{,}08 = \underbrace{0{,}08}_{\substack{nueva\\ mantisa}} + \underbrace{0}_{\substack{parte\ entera:\\ DÍGITO\ EN\ b_d}}$$

Se ha alcanzado el número máximo de bits fraccionarios disponibles (8).

La mantisa es (aproximada por truncamiento a 8 bits): $0{,}43_{(10} \approx 0{,}01101110_{(2}$

Se alcanza: $21{,}43_{(10} = 21_{(10} + 0{,}43_{(10} = 10101_{(2} + 0{,}01101110_{(2} = 10101{,}01101110_{(2}$

3.3. Cambio de base entre $b_o \neq 10$ y $b_d \neq 10$

Si tanto la base de origen b_o como la base de destino b_d son ambas distintas de base 10:

1. Se usa TFN para escribir los dígitos de la base de origen en base 10, que constituye un RELEVO intermedio.
2. Se usa Teorema del resto para alcanzar la base de LLEGADA, desde la secuencia en base 10.

EJEMPLO 5 : CAMBIO DE BASE ENTRE BASES DISTINTAS DE 10 APLICANDO TFN Y TEOREMA DEL RESTO SUCESIVAMENTE

Se desea expresar $321_{(4}$ en base 3.

Basta con aplicar sucesivamente los 2 mecanismos (coincide con los 2 ejemplos previos):

$$321_{(4} \underbrace{=}_{\substack{por\\TFN}} 57_{(10} \underbrace{=}_{\substack{por\ T^a\\RESTO}} 2010_{(3}$$

Es decir:

$$321_{(4} = 3 \cdot 4^2 + 2 \cdot 4^1 + 1 \cdot 4^0 = 57_{(10}$$

Y por último:

$$\underbrace{57_{(10}}_{\substack{secuencia\\en\ base_o}} = \underbrace{3}_{\substack{nueva\\base}} \cdot \underbrace{19}_{cociente} + \underbrace{0}_{resto}$$

$$\underbrace{19}_{\substack{cociente\\previo\\en\ base_o}} = \underbrace{3}_{\substack{nueva\\base}} \cdot \underbrace{6}_{cociente} + \underbrace{1}_{resto}$$

$$\underbrace{6}_{\substack{cociente\\previo\\en\ base_o}} = \underbrace{3}_{\substack{nueva\\base}} \cdot \underbrace{2}_{cociente} + \underbrace{0}_{resto}$$

$$\underbrace{2}_{\substack{cociente\\previo\\en\ base_o}} = \underbrace{3}_{\substack{nueva\\base}} \cdot \underbrace{0}_{cociente} + \underbrace{2}_{resto}$$

Como se desea calcular la parte ENTERA, se leen los dígitos de ABAJO HACIA ARRIBA (en ORDEN CONTRARIO al que aparecieron: el MÁS RECIENTE, tendrá MÁS PESO). Se alcanza:

$$321_{(4} \underbrace{=}_{\substack{por\\TFN}} 57_{(10} \underbrace{=}_{\substack{por\ T^a\\RESTO}} 2010_{(3}$$

EJEMPLO 6 : CAMBIO DE BASE APLICANDO TFN Y TEOREMA DEL RESTO SUCESIVAMENTE CON PARTE FRACCIONARIA

Se desea expresar $463{,}28_{(7}$ en base 5 con una precisión de 3 cifras fraccionarias.

1) Primero se hace un relevo en base 10, mediante TFN:

$$463{,}28_{(7} = 4 \cdot 7^2 + 6 \cdot 7^1 + 3 \cdot 7^0 + 2 \cdot 7^{-1} + 8 \cdot 7^{-2} = 241 + \frac{2}{7} + \frac{8}{49} = 241{,}448_{(10}$$

2) Ahora se expresa la secuencia en la base de llegada (base 5) mediante TEOREMA DEL RESTO:

Para la parte entera:

$$\underbrace{241}_{\substack{dígito \\ en\ base_o}} = \underbrace{5}_{\substack{nueva \\ base}} \cdot \underbrace{48}_{cociente} + \underbrace{1}_{resto}$$

$$\underbrace{48}_{\substack{cociente \\ previo}} = \underbrace{5}_{\substack{nueva \\ base}} \cdot \underbrace{9}_{cociente} + \underbrace{3}_{resto}$$

$$\underbrace{9}_{\substack{cociente \\ previo}} = \underbrace{5}_{\substack{nueva \\ base}} \cdot \underbrace{1}_{cociente} + \underbrace{4}_{resto}$$

$$\underbrace{1}_{\substack{cociente \\ previo}} = \underbrace{5}_{\substack{nueva \\ base}} \cdot \underbrace{0}_{cociente} + \underbrace{1}_{resto}$$

Como se desea calcular la parte ENTERA, se leen los dígitos de ABAJO HACIA ARRIBA (en ORDEN CONTRARIO al que aparecieron: el MÁS RECIENTE, tendrá MÁS PESO). Se alcanza:

$$463_{(7} = 1431_{(5}$$

Para la parte fraccionaria (hasta la precisión pedida de 3 cifras fraccionarias):

$$\underbrace{0{,}448}_{\substack{mantisa \\ original}} \cdot \underbrace{5}_{\substack{nueva \\ base}} = 2{,}24 = \underbrace{0{,}24}_{\substack{nueva \\ mantisa}} + \underbrace{2}_{\substack{parte\ entera: \\ DÍGITO\ EN\ b_d}}$$

$$\underbrace{0{,}24}_{\substack{mantisa \\ obtenida}} \cdot \underbrace{5}_{\substack{nueva \\ base}} = 1{,}2 = \underbrace{0{,}2}_{\substack{nueva \\ mantisa}} + \underbrace{1}_{\substack{parte\ entera: \\ DÍGITO\ EN\ b_d}}$$

$$\underbrace{0{,}2}_{\substack{mantisa \\ obtenida}} \cdot \underbrace{5}_{\substack{nueva \\ base}} = 1 = \underbrace{0}_{\substack{nueva \\ mantisa}} + \underbrace{1}_{\substack{parte\ entera: \\ DÍGITO\ EN\ b_d}}$$

Se alcanza:

$$463{,}28_{(7} = 241{,}448_{(10} = 1431{,}211_{(5}$$

Se puede verificar:

$$0{,}28_{(7} = \frac{2}{7} + \frac{8}{49} = \frac{22}{49} = 0{,}448_{(10}$$

$$0{,}211_{(5} = \frac{2}{5} + \frac{1}{25} + \frac{1}{125} = 0{,}448_{(10}$$

3.4. Gestión de los dígitos de la base de origen no disponibles en la base de destino

a) Si la base de origen es 10 (se usa teorema del resto) y la de destino es hexadecimal, se reescriben los dígitos de la base de origen mayores que 9 usando letras.
b) Si la base de origen es hexadecimal y la de destino es 10 (se usa TFN), se reescriben las letras usando sus valores (de 10 a 15).
c) Si ninguna de las 2 bases es 10, entre la aplicación del teorema fundamental de la numeración y la del teorema del resto, se reescriben los dígitos de la base de origen mayores que 9 usando letras.

EJEMPLO 7 : DE BASE 10 A BASE HEXADECIMAL

Se desea expresar $41407{,}12_{(10}$ en base 16 (hexadecimal):

Para la parte entera:

$$\begin{aligned} 41407 &= 16 \cdot 2587 + 15 \\ 2587 &= 16 \cdot 161 + 11 \\ 161 &= 16 \cdot 10 + 1 \\ 10 &= 16 \cdot 0 + 10 \end{aligned}$$

Los restos son los dígitos en la base de llegada, pero ni 15, ni 11 ni 10 existen en hexadecimal.
Se escriben en hexadecimal:

$15_{(10} = F_{(16}$ $\quad$ $11_{(10} = B_{(16}$ $\quad$ $10_{(10} = A_{(16}$

Entonces:

$$41407_{(10} = A1BF_{(16}$$

Para la parte fraccionaria:

$$\begin{aligned} 0{,}12 \cdot 16 &= 1{,}92 = 0{,}92 + 1 \\ 0{,}92 \cdot 16 &= 14{,}72 = 0{,}72 + 14 \\ 0{,}72 \cdot 16 &= 11{,}52 = 0{,}52 + 11 \\ 0{,}52 \cdot 16 &= 8{,}32 = 0{,}32 + 8 \\ 0{,}32 \cdot 16 &= 5{,}12 = 0{,}12 + 5 \\ 0{,}12 \cdot 16 &= 1{,}92 = 0{,}92 + 1 \\ 0{,}92 \cdot 16 &= 14{,}72 = 0{,}72 + 14 \end{aligned}$$

Se detiene al identificarse un comportamiento periódico (se omitirán los últimos 2 dígitos por ser redundantes).
Se escribe en hexadecimal cada resto que excede 10 (no existen como tal en base 16):

$$14_{(10} = E_{(16}$$
$$11_{(10} = B_{(16}$$

Se alcanza:

$$0{,}12_{(10} = 0{,}1\widehat{EB}85_{(16}$$

Uniendo parte entera y mantisa, se obtiene:

$$41407{,}12_{(10} = A1BF{,}1\widehat{EB}85_{(16}$$

3.5. Cambio de base entre potencias de la misma base

a) Cambio de base desde b hacia b^n

Cada dígito en una base b^n corresponde con exactamente n dígitos en base b.
Por ejemplo, cada dígito en base 8 corresponde con 3 dígitos en base 2, ya que $8 = 2^3$.

1. Cada dígito en base b^n corresponde con n dígitos de la secuencia en base b.
2. Se agrupan los dígitos de la secuencia de llegada de n en n desde la coma fraccionaria (ya sea hacia la parte entera o hacia la parte fraccionaria).
3. Si las agrupaciones terminales quedan incompletas, se completan con 0 SIEMPRE POR LA IZQUIERDA, tanto para la parte entera, como para la parte fraccionaria.
4. Se expresa cada agrupación de la secuencia de salida en la base de llegada.
5. Se unen las secuencias resultantes.

EJEMPLO 8 : CAMBIO DE BASE ENTRE b Y b^n

Se desea expresar la secuencia $01101010,0101_{(2}$ en base 16.

Nótese que $16 = 2^4$ y, por tanto, CADA DÍGITO en base 16 equivale a 4 dígitos en base 2.

O sea, la secuencia de salida 0110 1010 , $0101_{(2}$ (de 12 dígitos en base 2) tiene 4 grupos de 4 dígitos en base 16. Los grupos se toman:

- Desde la coma fraccionaria hacia la izquierda para la parte entera
- Desde la coma fraccionaria hacia la derecha para la parte fraccionaria

Pero en ambos casos, se completan por la izquierda con 0 en caso de que el último en formarse, no tenga exactamente 4 dígitos.

Se toman los 3 grupos en la base de salida que corresponden con los 3 dígitos en la base de llegada, cada uno de los cuales formado por 4 dígitos

$0110_{(2}$ $1010_{(2}$ $0101_{(2}$

Y se expresan en la base de llegada:

$0110_{(2} = 4 + 2 = 6_{(10} = 6_{(16}$
$1010_{(2} = 8 + 2 = 10_{(10} = A_{(16}$
$0101_{(2} = 4 + 1 = 5_{(10} = 5_{(16}$

Es decir: $01101010,0101_{(2} = 6A,5_{(16}$

EJEMPLO 9 : CAMBIO DE BASE ENTRE b Y b^n COMPLETANDO GRUPOS

Se desea expresar $101110{,}101101_{(2}$ en base 16.

Se agrupan de 4 en 4 ya que $16 = 2^4$ y, por tanto, cada dígito en base 16 son 4 dígitos en base 2

Se agrupan DESDE LA COMA FRACCIONARIA, completando con 0 SIEMPRE POR LA IZQUIERDA hasta tener 4 dígitos por grupo.

Parte entera:

Se tiene:	101110
Se divide desde la come en 2 grupos de 4:	10 1110
Se completa el grupo final por la IZQUIERDA:	0010 1110
Ahora, por TFN, se escriben en base 16:	$0010_{(2} = 2_{(16}$
	$1110_{(2} = 8 + 4 + 2 = 14_{(10} = E_{(16}$

Parte fraccionaria:

Se tiene:	101101
Se divide desde la come en 2 grupos de 4:	1011 01
Se completa el grupo final por la IZQUIERDA:	1011 0001
Ahora, por TFN, se escriben en base 16:	$1011_{(2} = 8 + 2 + 1 = 11_{(10} = B_{(16}$
	$0001_{(2} = 1_{(16}$

Se unen:

$$101110{,}101101_{(2} = 2E, B1_{(16}$$

b) Cambio de base desde b^n hacia b

Cada dígito en una base b^n corresponde con exactamente n dígitos en base b.
Por ejemplo, cada dígito en base 8 corresponde con 3 dígitos en base 2, ya que $8 = 2^3$.

Basta con tomar cada uno de los dígitos de la base de origen b^n y expresarlos en la base de destino b mediante TFN.
Luego, se completa con 0 (SIEMPRE POR LA IZQUIERDA tanto para la parte entera como para la parte fraccionara) la secuencia resultante de la conversión de cada dígito hasta alcanzar longitud n.

EJEMPLO 10 : CAMBIO DE BASE ENTRE b^n Y b COMPLETANDO GRUPOS

Se desea expresar la secuencia $673{,}14_{(8}$ en base 2.

Como $2^3 = 8$ se ven grupos de 3 dígitos en la base de destino 2 para codificar cada 1 de los dígitos de la base de origen 8:

6	7	3	,	1	4	Base 8
110	111	011	,	001	100	grupos de longitud 3 en base 2

$$673{,}54_{(8} = 110\ 111\ 011{,}001\ 100$$

3.6. Resumen de estrategias de cambio de base

Si la base de DESTINO es 10, basta con aplicar TFN.

Si la base de ORIGEN es 10, se aplica el Teorema del resto para alcanzar la base de llegada:

- Si se calcula parte ENTERA — Los dígitos MÁS RECIENTES son los de MÁS PESO.
- Si se calcula una parte FRACCIONARIA — Los dígitos MÁS RECIENTES son los de MENOS PESO.

Si la base de origen y la de destino son distintas de 10 y ninguna es potencia de la otra:

1. Se usa TFN para expresar la secuencia en base 10.
2. Se gestionan los dígitos no admisibles en base 10.
3. Se usa el TEOREMA DEL RESTO para alcanzar la secuencia en base de destino.

Si la base de DESTINO ES POTENCIA de la de origen:

Se agrupan los dígitos de la base de origen b_o en grupos de longitud m equivalente a la potencia de la base de origen que es la base de destino $b_d = b_o^m$.

El agrupamiento tiene 2 requisitos:

- Se hace desde la posición de la coma fraccionaria, hacia la izquierda para la parte entera y hacia la derecha para la parte fraccionaria.
- Si se alcanza el último grupo, el más alejado de la coma (tanto en la parte entera como en la fraccionaria), se COMPLETA CON 0 hasta conseguir el mismo número de dígitos que en resto de grupos. Se completa en el MISMO SENTIDO DE LECTURA: dese la coma y hacia los extremos
    - Se completa por la IZQUIERDA en la parte ENTERA.
    - Se completa por la DERECHA en la parte FRACCIONARIA

Se aplica TFN para escribir cada grupo de dígitos de la base de origen como dígito único en la base de destino.

Si la base de ORIGEN ES POTENCIA de la de destino:

Se aplica TFN para escribir cada dígito de la base de origen como grupo de dígitos de longitud n en la base de destino.
Para completar grupos, en la parte entera se hace por la izquierda, mientras que, en la parte fraccionaria, se añaden 0 por la derecha. O sea, de forma simétrica desde la coma.

3.7. ERROR COMÚN: dígitos inexistentes en la base de origen

Para que una secuencia tenga sentido expresada en una base b, todos los dígitos que la componen deben pertenecer a los que la base ofrece.

EJEMPLO 11 : EXCESO DE LOS DÍGITOS EXISTENTES

Se desea expresar $468_{(7}$ en base 5.

La base 7 no contiene el dígito 8, por tanto, $468_{(7}$ es un absurdo y no se puede expresar en ninguna otra base.

4. Empaquetamiento hexadecimal

No es un cambio de base.
Se aplica a secuencias binarias (expresadas en base 2).
Se denota con una letra h al final de la secuencia obtenida.

Para el empaquetamiento hexadecimal de una secuencia binaria:

1. Se omite la coma fraccionaria de la secuencia original.
2. Se agrupa la secuencia binaria en grupos de cuatro bits, empezamos desde la derecha.
3. Se completa el último grupo hasta alcanzar los cuatro bits.
4. Se escribe, mediante TFN, cada 1 de los grupos en base 16.

EJEMPLO 12 : empaquetamiento hexadecimal vs. Cambio de base

Se desea empaquetar en hexadecimal la secuencia binaria 11001011,101

Se omite la coma:	11001011101		
Se hacen grupos de 4 bits:	110	0101	1101
Se completa el último:	0110	0101	1101
Se escriben (por TFN) en hexadecimal:	6	5	D

Se alcanza: $11001011101_{(2}$ = 65Dh

Por contrario, el cambio de base sí contempla la coma fraccionaria para hacer los grupos:

Se agrupan desde la coma en grupos de 4:	1100	1011	,101	
Se completan por la izquierda con 0:		1100	1011	,0101
Se escriben en base 16:	C	B	5	

Se alcanza: $11001011101_{(2}$ = $CB,5_{(16}$

5. Representación de cantidades en computadores y error de representación

La capacidad de representación de cantidades que tiene un computador es finita, de modo que no permite representar números con partes fraccionarias infinitas sin cometer error.

El error ε en la representación es la diferencia entre el número X que se desea representar y el que se ha representado $\hat{X}$:

$$\varepsilon = |X - \hat{X}|$$

6. Rango de representación

El rango de representación es un intervalo que contiene los valores representables dado un formato:

$$rango = [mínimo, máximo]$$

Nótese que:

- Se trata de un intervalo cerrado en que los extremos están incluidos.
- No solo depende del número de dígitos escogido, sino también de la especificación del formato.

7. Precisión de representación

La precisión de representación depende del formato de representación.

La precisión es la separación mínima que el formato es capaz de representar entre 2 valores, o sea, es proporcional al número de posiciones fraccionarias que admite.

La precisión de un formato $x_0, x_{-1}x_{-2}$ es de 0,01 mientras que para un formato como x_1x_0 es de 1.

8. Aproximación: Truncamiento y redondeo

En este curso se emplean 2 métodos de aproximación para la gestión del error de representación:

- Truncamiento
- Redondeo

8.1. Truncamiento

Consiste en omitir los dígitos fraccionarios más allá de la posición que especifique el formato.

EJEMPLO 13 : APROXIMACIÓN POR TRUNCAMIENTO
Se desea aproximar por truncamiento $10{,}123456_{(10}$ a un formato con exactamente dos cifras fraccionarias.

El formato escogido impone 2 dígitos fraccionarios y se considera una cantidad cuya representación exacta requiere de 6 dígitos fraccionarios. Para TRUNCARLO, los 4 dígitos de menor peso se desprecian. En un formato $x_1x_0, x_{-1}x_{-2}$ la representación de la cantidad $10{,}123456_{(10}$ es 10,12 y el resto de las cifras se descarta.

8.2. Redondeo

Aprovecha el concepto de precisión para mejorar la exactitud del truncamiento.
Dado un número a representar, permite encontrar el número más cercano representable en el formato especificado.

Para ello:

1) Se considera la precisión del formato especificado.
2) Se calcula la mitad de la precisión.
3) Se suma la mitad de la precisión al número a representar.
4) Se trunca el resultado de acuerdo con el número de dígitos fraccionarios que especifica el formato.

EJEMPLO 14 : APROXIMACIÓN POR REDONDEO

Se desea aproximar por redondeo $1{,}23456_{(10}$ en el formato de representación $x_1x_0,x_{-1}x_{-2}x_{-3}$

1) Se identifica la precisión DEL FORMATO (no del número dado):
   Con 3 cifras fraccionarias, la precisión es de $0{,}001_{(10}$

2) Se calcula la mitad de la precisión:

$$0{,}001 : 2 = 0{,}0005_{(10}$$

3) Se suma a la cantidad a redondear, la mitad de la precisión:

$$1{,}23456_{(10} + 0{,}0005_{(10} = 1{,}23506_{(10}$$

4) Se trunca a 3 cifras fraccionarias el resultado:

$$1{,}23506_{(10} \rightarrow 1{,}235_{(10}$$

EJEMPLO 15 : APROXIMACIÓN POR REDONDEO EN BASE 2

Se desea aproximar por redondeo $10{,}10011_{(2}$ en el formato de representación $x_1x_0,x_{-1}x_{-2}x_{-3}$

La precisión es: $0{,}001_{(2}$

La mitad de la precisión es: $0{,}001_{(2} : 2_{(10} = 0{,}001_{(2} : 10_{(2} = 0{,}0001_{(2}$

O sea, en base 2, la división entre 2 se puede ver como el desplazamiento de 1 dígito hacia la izquierda.

Se suma la mitad de la precisión al número a redondear:

$$10{,}10011_{(2} + 0{,}0001_{(2} = 10{,}10101_{(2}$$

Se trunca el resultado a 3 cifras fraccionarias:

$$10{,}10101_{(2} \rightarrow 10{,}101_{(2}$$

9. OPERACIONES EN SISTEMAS POSICIONALES

9.1. Suma en sistemas posicionales: BINARIO y ACARREO

El algoritmo ordinario de la suma es válido para cualquier sistema posicional, con independencia de la base empleada. O sea, el acarreo que resulta en una posición se incorpora como operando en la suma de la posición siguiente.

Para gestionar el acarreo, el resultado de una suma de 2 bits en base 2, se codifica utilizando dos campos separados:

- El bit de resultado.
- El bit de transporte, también denominado “carry” (del inglés, para lo que habitualmente se llama “me llevo una…”)

La suma en binario, con 2 operandos A y B, se puede ver como:

A	+	B	=	Carry	Resultado
0	+	0	=	0	0
1	+	0	=	0	1
0	+	1	=	0	1
1	+	1	=	1	0

Además, con transporte procedente de una operación previa:

Acarreo previo	A	+	B	=	Carry	Resultado
1	0	+	0	=	0	1
1	1	+	0	=	1	0
1	0	+	1	=	1	0
1	1	+	1	=	1	1

En conclusión, según la cantidad de 1 que presenta la columna que se opera:

- Si no hay ninguno carry = 0 resultado = 0
- Si hay 1 carry = 0 resultado = 1
- Si hay 2 carry = 1 resultado = 0
- Si hay 3 (acarreo previo) carry = 1 resultado = 1

EJEMPLO 16 : SUMA EN BINARIO NATURAL

1	1	1		1						acarreo
	1	0	1	0	1	0	1	0	(2	A
+	1	1	1	0	1	1	0	0	(2	B
1	1	0	0	1	0	1	1	0	(2	resultado A + B

EJEMPLO 17 : SUMA EN HEXADECIMAL SIN ACARREO

Un error común es operar los valores en decimal guiados por el carácter intuitivo del cambio de base:

$A_{(16} = 10_{(10}$ $B_{(16} = 11_{(10}$ $C_{(16} = 12_{(10}$
$D_{(16} = 13_{(10}$ $E_{(16} = 14_{(10}$ $F_{(16} = 15_{(10}$

Pero en realidad, se cumple:

								← acarreo
		6	5	4	3	2	1	(16	
	+	9	9	9	9	9	9	(16	
		F	E	D	C	B	A	(16	← resultado

EJEMPLO 18 : SUMA EN HEXADECIMAL CON ACARREO

	1	1	1					← acarreo
			1	3	3	2	1	(16	
	+	F	E	D	C	B	A	(16	
	1	0	0	0	F	D	B	(16	← resultado

O sea:

$1 + A = B$ $B + 2 = D$ $3 + C = F$ $3 + D = 10$
$1 + 1 + E = 10$ $1 + F = 10$

9.2. Resta en sistemas posicionales

El algoritmo ordinario de la resta es válido para cualquier sistema posicional, con independencia de la base empleada. O sea, el acarreo que resulta en una posición se incorpora como operando en la suma de la posición siguiente.

Para gestionar el acarreo, el resultado de una suma de 2 bits en base 2, se codifica utilizando dos campos separados:

- El bit de resultado.
- El bit de transporte, también denominado "borrow" (del inglés, para lo que habitualmente se llama "tomo prestado una…")

La resta en binario, con 2 operandos A y B, se puede ver como:

A	-	B	=	Borrow	Resultado	Se puede ver como:
0	-	0	=	0	0	
1	-	0	=	0	1	
0	-	1	=	1	1	10 – 1 = 1 y me llevo 1
1	-	1	=	0	0	

Con transporte previo, es útil verlo como un sustraendo más (o sea, A – carry – B en cada columna):

A	-	Borrow	-	B	=	Borrow	Resultado	Se puede ver como:
0	-	1	-	0	=	1	1	10 – 1 = 1, me llevo 1. Después 1 – 0 = 1
1	-	1	-	0	=	0	0	
0	-	1	-	1	=	1	0	10 – 1 = 1 y me llevo 1. Después 1 – 1 = 0 O bien en base 10: 2 – 2 = 0
1	-	1	-	1	=	1	1	1 – 1 = 0. Después 10 – 1 = 1 y me llevo 1 O bien en base 10: 3 – 2 = 1

O sea, es conveniente considerar A – B asociativamente, en cada columna: (A – borrow) – B

EJEMPLO 19 : RESTA EN BASE 2

	1	0	0	1	0	0	1	1	0	A	← minuendo
1	1	1		1	1						← borrow
-	1	1	1	0	0	1	1	0	0	B	← sustraendo
1	1	0	1	0	1	1	0	1	0		← resultado

10. Desbordamiento

Si el resultado de una operación aritmética excede el rango de representación del formato escogido para operar, se dice que se produce desbordamiento.

EJEMPLO 20 : DESBORDAMIENTO

Se consideran las secuencias A = $01100110_{(2}$ y B = $11010101_{(2}$ expresadas en binario natural con un formato de 8 bits.

Se desea saber si la operación A + B genera desbordamiento.

1	1				1				acarreo
	0	1	1	0	0	1	1	0 (2	A
+	1	1	0	1	0	1	0	1 (2	B
desbordamiento	0	0	1	1	1	0	1	1 (2	resultado A + B

Se observa desbordamiento.

El desbordamiento compromete la corrección del resultado de una operación.

11. Condición de desbordamiento según el formato:

- En binario natural, hay desbordamiento cuando hay acarreo operando el bit de mayor peso.

- En signo y magnitud (como en binario natural), hay desbordamiento si se observa acarreo como resultado de operar los bits de mayor peso, de modo que no hay suficientes bits en el formato especificado para representar correctamente el resultado.

- En complemento a 2, hay desbordamiento si el signo del resultado es opuesto al signo que se espera, de modo que su evaluación exige un análisis previo para luego contrastar lo que se obtiene con lo que se esperaba.

12. Números naturales: rango, precisión y extensión

Los números naturales no tienen signo ni tampoco parte fraccionaria.

El 0 se considera natural en este curso.

En binario, para una longitud de n bits se tiene un rango de representación:

$$[0,2^n - 1]$$

La precisión que ofrece el conjunto de los números naturales es 1.

Para la extensión de bits de números naturales, basta con añadir 0 a la izquierda de la secuencia: $1011_{(2}$ con 4 bits se escribe como $001011_{(2}$ con 6 bits.

13. Números con signo

Para identificar magnitudes negativas y distinguirlas de las positivas, se emplean los signos – y + respectivamente:

$-1011_{(2}$ $+C5{,}4D_{(16}$

14. Formatos de representación

En los computadores, la especificación concreta de la representación de un número se denomina formato de representación.

En este curso, se trabaja con los formatos:

- Para números NATURALES	Binario natural	$9_{(10} = 1001_{(2}$
- Para números ENTEROS	Signo y magnitud	$-9_{(10} = 11001_{(SM}$
	Complemento a 2	$-9_{(10} = 10111_{(Ca2}$
- Para números FRACCIONARIOS	De coma fija	$9{,}25_{(10} = 1001{,}01$
	De coma flotante	

15. Números enteros

Se entiende por número entero aquel que cumple 2 requisitos:

- Sí tiene signo (se distinguen negativos de positivos).
- Su parte fraccionaria es nula.

15.1. Formato Signo y magnitud (SM)

a) Especificación

La especificación de un sistema de representación es la definición del formato o esquema que lo define. El formato signo y magnitud es un esquema de representación de números en base 2.

Se basa en reservar el bit más significativo de la secuencia para la codificación del signo, mientras que el resto de la secuencia codifica la magnitud representada:

- Si el bit más significativo es 0 la magnitud representada es positiva
- Si el bit más significativo es 1 la magnitud representada es negativa

Se denota por el subíndice SM: $1101_{(SM} = -101_{(2} = -5_{(10}$

En la secuencia se identifican 2 compartimentos: $\underbrace{1}_{bit\ de\ signo}\ \underbrace{101}_{magnitud}$

Para la especificación completa del formato hay que definir el número de bits disponibles:

$$-20_{(10}\ codificado\ en\ SM\ con\ 6\ bits\ es\ 110100_{(SM}$$

Se ve el bit de mayor peso para codificar el signo y los otros 5 para la magnitud:

$$\underbrace{1}_{<0}\ \underbrace{10100}_{|20_{(10}|}$$

b) Rango de representación en signo y magnitud en base 2: el "0 doble"

El rango de representación en signo y magnitud en base dos es, para una secuencia de n bits:

$$[-(2^{n-1}-1), 2^{n-1}-1]$$

Es decir, es un rango de representación simétrico respecto la frontera entre positivos y negativos, No obstante, incurre en una incongruencia grave: La secuencia que codifica el valor 0 está repetida.

Por ejemplo, con 2 bits: $00_{(SM} = +0_{(10}$ $10_{(SM} = -0_{(10}$

Así hay aparente "0 positivo" y un aparente "0 negativo", lo cual es relativamente rudimentario.

c) Suma y resta en signo y magnitud

Se distinguen dos casos para sumar y restar en signo de magnitud:

- Si los dos números a operar son del mismo signo:
  - Sus magnitudes se suman.
  - El resultado hereda el mismo signo de los operandos

  Esto es así con independencia de si son positivos o negativos
  Para operar, basta con sumar las magnitudes y conservar el signo.

- Si los dos números a operar son de signo opuesto:
  - Sus magnitudes se restan
  - El resultado hereda el signo de la magnitud mayor

  Se analiza cuál de las dos magnitudes es mayor.
  Se resta la menor de la mayor

EJEMPLO 21 : SUMA DE 2 NÚMEROS NEGATIVOS EN SM

Se tienen las secuencias codificadas en SM y 6 bits:

$A = 101000_{(SM}$ y $B = 101010_{(SM}$

Se desea calcular A + B.

Para ello, se comprueba que son del mismo signo: $A < 0$ y $B < 0$ puesto que empiezan por 1.

Se toman las magnitudes: $|A| = 01000$ y $|B| = 01010$

Se suman las magnitudes:

	1					acarreo
	0	1	0	0	0	|A|
+	0	1	0	1	0	|B|
	1	0	0	1	0	|A+B|

Se añade a la magnitud |A+B| el bit de signo que hereda de los operandos:

$$A + B = \underbrace{1}_{<0}\ \underbrace{10010}_{|A+B|}{}_{(SM}$$

EJEMPLO 22 : SUMA DE 2 NÚMEROS DE SIGNO OPUESTO EN SM

Se tienen las secuencias codificadas en SM y 6 bits:
$A = 101000_{(SM}$ y $B = 001010_{(SM}$
Se desea calcular A + B.

Se tiene:

$$A < 0 \; ya \; que \; su \; bit \; más \; significativo \; es \; S = 1$$
$$B > 0 \; ya \; que \; su \; bit \; más \; significativo \; es \; S = 0$$

Como su signo es opuesto, hace falta determinar el signo del resultado A + B.
Para ello, se analiza cuál de las dos magnitudes es mayor en valor absoluto:

$$|A| = |01000|_{(2} = 8_{(10}$$
$$|B| = |01010|_{(2} = 10_{(10}$$

$$|B| > |A| \rightarrow signo(A+B) = signo(B) \rightarrow A + B > 0$$

Es decir, el resultado es POSITIVO.
Y ahora, para definir la magnitud del resultado, se sustrae la menor magnitud de la mayor:

$$|A+B| = |B| - |A|$$

En este caso:

	0	1	0	1	0	|B|
						borrow
-	0	1	0	0	0	|A|
	0	0	0	1	0	|A+B|

O sea:

$$A + B = 010_{(SM}$$

En signo y magnitud SM, para sumar números de signo opuesto, se opera una resta.

d) Extensión de bits en binario natural y SM

La extensión de una secuencia es la modificación del formato especificado a través de la adición de ceros por la izquierda, lo cual permite la ampliación de su rango de representación.

EJEMPLO 23 : EXTENSIÓN DE BITS
Se desea extender la secuencia $101_{(2}$ en binario natural hasta que se encuentre en un formato cuyo rango sea [0,31].

Con 3 bits de longitud, el rango es: $[0,2^3 - 1]$ o sea, [0,7] en base 10.
El rango [0,31] corresponde con $[0,2^6 - 1]$ o sea, que se requieren 6 bits.
Por tanto, hace falta extender en 3 bits la secuencia dada:

$$101_{(2} \rightarrow 000101_{(2}$$

15.2. Formato Complemento a 2 (Ca2)

El complemento a dos es el formato de representación mayoritario en los computadores.
Se denota por Ca2.

Permite la representación de números enteros (con signo).

Es relevante notar que, puesto que en Ca2 se representan ENTEROS, el RANGO sufre una merma, puesto que el mismo número de dígitos que en binario natural sirva para alejarse del 0 en 1 solo sentido, en Ca2 lo hace en AMBOS sentidos.

a) Especificación Ca2

En signo y magnitud SM, cada dígito contribuye a hacer la magnitud mayor (cada bit no nulo que se añade a la secuencia aleja el número representado del 0, o sea, incrementa su valor absoluto, ya sea en sentido positivo o negativo).

En cambio, la construcción de valores en Complemento a 2 es "sustractiva": el bit de mayor peso es siempre negativo (tanto si es 0 como si es 1), y el resto son positivos.

Es decir, en Ca2 la información de signo no está separada de la información de magnitud, como en SM, sino que el primer bit codifica tanto signo como magnitud.

Igual que en SM, en Ca2, una secuencia que comienza por 1 representa un número negativo, mientras que si empieza por 0 representa un número positivo.

EJEMPLO 24 : CODIFICACIÓN EN COMPLEMENTO A 2

$$-10_{(10} = -16 + 4 + 2 = -2^4 \cdot 1 + 2^3 \cdot 0 + 2^2 \cdot 1 + 2^1 \cdot 1 + 2^0 \cdot 0 = 10110_{(Ca2}$$

$$10_{(Ca2} = -2^4 \cdot 0 + 2^3 \cdot 1 + 2^2 \cdot 0 + 2^1 \cdot 1 = 0101_{(Ca2}$$

Mientras que en SM la secuencia que codifica un número coincide con la secuencia que codifica su opuesto (el número negativo con el mismo valor absoluto) y solo los diferencia el bit de signo (0 paras el positivo y 1 para el negativo), en Ca2 las secuencias de números opuestos no coindicen.

Lo más característico de la codificación en Complemento a 2 es lo referente a los números negativos: el primer dígito tiene un valor posicional siempre negativo (de modo que cuando vale 1 computa y cuando vale 0 no contribuye) mayor que la suma de los valores de todos los demás dígitos de menor peso.

Por tanto, el mecanismo mediante el cual el formato de Ca2 produce un número negativo se basa en sustraer al valor que codifica el bit de mayor peso la suma de los valores que codifican el resto, que son siempre positivos.

b) Números positivos en Ca2

La secuencia que codifica un número positivo es la misma expresada en Ca2 y en SM.

EJEMPLO 25 : CODIFCIACIÓN EN Ca2 DE UN NÚMERO POSITIVO

Se desea representar el número $+5_{(10}$ en SM y en Ca2.

En SM:

$$+5_{(10} = 0101_{(SM}$$

En Ca2:

$$+5_{(10} = -2^3 \cdot 0 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 0101_{(Ca2}$$

Nótese que en binario natural bastan 3 dígitos para representar $+5_{(10}$

$$+101_{(2} = +5_{(10}$$

El dígito adicional presente en las secuencias codificadas en SM y Ca2 responde a la necesidad de incorporar el signo.

c) Números negativos en Ca2

Un número negativo no presenta la misma secuencia en signo y magnitud que en Ca2.

EJEMPLO 26 : CODIFCIACIÓN EN Ca2 DE UN NÚMERO NEGATIVO vs. EN SM

Se desea representar el número $-5_{(10}$ en SM y en Ca2.

En SM:

$$-5_{(10} = 1101_{(SM}$$

En Ca2:

$$-5_{(10} = -2^3 + 2^1 + 2^0 = 1011_{(Ca2}$$

Se distinguen 3 formas para calcular la secuencia que codifica un número negativo en Ca2.

1) Usando TFN y considerando el bit de mayor peso negativo (por "sustracción")

EJEMPLO 27 : USO DE TFN PARA CODIFICAR EN Ca2
Se desea codificar en Ca2 el valor $-18_{(10}$
Para ello, se toma la primera potencia de 2 que excede la magnitud deseada y se escribe con signo negativo. a continuación, se aproxima la magnitud hacia el 0 hasta el valor deseado, mediante la suma estratégica de otras de potencias de 2 de signo positivo.

$$-18_{(10} = -32 + 8 + 4 + 2 = -2^5 + 2^3 + 2^2 + 2^1 = 101110_{(Ca2}$$

EJEMPLO 28 : USO DE TFN PARA CODIFICAR EN Ca2

$$-26_{(10} = -32 + 4 + 2 = -2^5 \cdot 1 + 2^2 + 2^1 = 100110_{(Ca2}$$

Es relevante notar que en Ca2, una secuencia que codifica un número negativo representa un mayor valor absoluto (más alejada del cero por la izquierda) cuanto más poblada de 0 está:

$$|1111_{(Ca2}| < |1000_{(Ca2}|$$

$$|-1_{(10}| < |-8_{(10}|$$

2) Mediante la definición formal:

$$X \text{ es un número binario de n bits del cual se desea conocer su } Ca2$$

$$X_{(Ca2} = 2^n - |X|$$

EJEMPLO 29 : USO DE DEFINICIÓN FORMAL DE Ca2 PARA CODIFICAR NÚMERO ENTERO NEGATIVO

Se desea expresar $-26_{(10}$ en Ca2 usando 6 bits.

1. Se tiene la magnitud sin signo: $|26|_{(10} = 16 + 8 + 2 = 11010_{(2}$

   La longitud de estas secuencias de 5 bits.

2. Se calcula $2^n = 2^6$ en binario: $2^6_{(10} = 100000_{(2}$

   La longitud de esta secuencia es de 7 bits.

3. Para poder operar ambas secuencias, su longitud debe coincidir. Para ello, se extiende la más corta hasta los 7 bits:

   $$11010_{(2} \rightarrow 0011010_{(2}$$

4. Se opera:

	1	0	0	0	0	0	0	2^6
	1	1	1	1	1			borrow
-	0	0	1	1	0	1	0	$\|26\|_{(2}$
	0	1	0	0	1	1	0	$-26_{(Ca2}$

   El “borrow” se interpreta como un dígito más a restar a la siguiente columna (análogamente al caso del “carry” en la suma, que se añade a la siguiente columna).

5. Por último, se restringe la longitud al número de bits deseado (6) omitiendo aquellos que excedan la posición sexta x_5:

   $$-26_{(Ca2} = 100110$$

3) Aprovechando el complemento a 1:

El complemento a 1 o complemento bit a bit consiste en encontrar el negado de cada bit, intercambiando 0 por 1 y viceversa. Se denota por NOT.

A partir de una secuencia X, se puede alcanzar su Ca2 como:

$$X_{(Ca2} = NOT(X) + 1$$

EJEMPLO 30 : USO DE Ca1 PARA CODIFICAR NÚMERO ENTERO NEGATIVO EN Ca2

Se desea expresar $-26_{(10}$ en Ca2 usando 6 bits.

$$|26_{(10}| = 11010_{(2}$$

Es NECESARIO extender la secuencia hasta el número de bits deseados ANTES de calcular su negado:

$$|26_{(10}| = 11010_{(2} \rightarrow 011010_{(2}$$

Se calcula el negado de la secuencia que corresponde a la magnitud de longitud deseada:

$$NOT(011010) = 100101_{(2}$$

Se añade 1 al resultado:

$$100101_{(2} + 1_{(2} = 100110_{(Ca2}$$

Se comprueba:

$$100110_{(Ca2} = -2^5 + 2^2 + 2^1 = -32 + 4 + 2 = -26_{(10}$$

d) Suma en Ca2

Se distinguen diversos casos según el signo de los operandos.
Para evaluar si se produce desbordamiento, no se considera el acarreo del bit más significativo (como es el caso de suma en SM) sino que se debe evaluar el signo del resultado obtenido y cotejarlo con el signo del resultado esperado.
Por tanto, antes de operar conviene considerar el signo del resultado esperado.

EJEMPLO 31 : SUMA DE 2 NÚMEROS POSITIVOS EN CA2 SIN DESBORDAMIENTO
Se tiene:

$$A = 0100_{(Ca2}$$
$$B = 0011_{(Ca2}$$

Se desea conocer A + B
Se espera un resultado positivo, o sea $A + B > 0$ ya que $A > 0, B > 0$
No obstante, hay riesgo de desbordamiento, puesto que necesariamente:

$$Si\ A > 0, B > 0 \rightarrow |A + B| > |A|, |B|$$

Se opera:

					acarreo
	0	1	0	0	$A_{(Ca2}$
+	0	0	1	1	$B_{(Ca2}$
	0	1	1	1	A+B

Se observa un resultado positivo, puesto que la secuencia empieza por 0, de modo que no se ha producido desbordamiento y el resultado puede ser correcto.

EJEMPLO 32 : SUMA DE 2 NÚMEROS POSITIVOS EN CA2 CON DESBORDAMIENTO
Se tiene:

$$A = 0101_{(Ca2}$$
$$B = 0111_{(Ca2}$$

Se desea conocer A + B
Se espera un resultado positivo, o sea $A + B > 0$ ya que $A > 0, B > 0$
No obstante, hay riesgo de desbordamiento, puesto que necesariamente:

$$Si\ A > 0, B > 0 \rightarrow |A + B| > |A|, |B|$$

Se opera:

	1	1	1		acarreo
	0	1	0	1	$A_{(Ca2}$
+	0	1	1	1	$B_{(Ca2}$
	1	1	0	0	A+B

Se observa resultado negativo, pues la secuencia empieza por 1, así que se confirma el desbordamiento.
EJEMPLO 33 : SUMA DE NÚMEROS DE DISTINTO SIGNO EN CA2

Se tiene:

$$A = 1100_{(Ca2}$$
$$B = 0011_{(Ca2}$$

Se desea conocer A + B

Para determinar el signo esperado para el resultado hay que evaluar las magnitudes operadas:

$$A = -2^3 + 2^2 = -4$$
$$B = 2^1 + 2^0 = 3$$

Se espera un resultado negativo, ya que $A < 0, B > 0$ y se ve $|A| > |B|$

Nótese que con operandos de signo opuesto nunca hay riesgo de desbordamiento:

$$Si\ signo(A) \neq signo(B) \rightarrow |A + B| \leq |A|, |B|$$

Se opera:

					acarreo
	1	1	0	0	$A_{(Ca2}$
+	0	0	1	1	$B_{(Ca2}$
	1	1	1	1	A+B

No hace falta evaluar el desbordamiento.
Se verifica el resultado en base 10:

$$A + B = -4 + 3 = -1_{(10}$$
$$1111_{(Ca2} = -8 + 4 + 2 + 1 = -1_{(10}$$

Es llamativo que, en Ca2, una secuencia que codifica un número negativo se encuentra más poblada de dígitos 1 cuanto más cerca del 0 se encuentra.

Esto contradice la idea intuitiva que sí cumple la codificación de números positivos es que una secuencia de bits codifica un número de mayor valor absoluto cuantos más dígitos 1 presenta.

e) Resta en Ca2

Puesto que el formato Ca2 permite la representación de números negativos, tanto la suma como la resta se opera como una suma, ya que el operando negativo se puede representar mediante la propiedad del opuesto Op:

$$A - B = A + Op(B)$$

Desde esta perspectiva, cada resta se puede ver como uno de los casos anteriores de suma:

Con $A > 0$:

$$Si\ B > 0 \rightarrow Op(B) < 0 \rightarrow Nunca\ hay\ riesgo\ de\ desbordamiento$$

Con	$\|A\| > \|B\|$	Se verá resultado positivo:	A – B > 0
Con	$\|A\| < \|B\|$	Se verá resultado negativo:	A – B < 0

$$Si\ B < 0 \rightarrow Op(B) > 0 \rightarrow Hay\ riesgo\ de\ desbordar$$

$A - B = A + Op(B) \rightarrow |A - B| > |A|, |B|$ Se espera resultado positivo

Con $A < 0$:

$$Si\ B < 0 \rightarrow Op(B) > 0 \rightarrow Nunca\ hay\ riesgo\ de\ desbordamiento$$

Con	$\|A\| > \|B\|$	Se verá resultado negativo:	A – B < 0
Con	$\|A\| < \|B\|$	Se verá resultado positivo:	A – B > 0

$$Si\ B > 0 \rightarrow Op(B) < 0 \rightarrow Hay\ riesgo\ de\ desbordar$$

$A - B = A + Op(B) \rightarrow |A - B| > |A|, |B|$ Se espera resultado negativo

EJEMPLO 34 : RESTA EN Ca2

Se tiene: $A = 011010_{(Ca2}$ $B = 001011_{(Ca2}$
Se desea conocer A – B.

Como Ca2 permite calcular el opuesto: $A - B = A + Op(B)$

Se espera un resultado positivo, ya que:

$$|A| > |B|, |B| = |Op(B)| \rightarrow |A| > |Op(B)|$$

Con A > 0 y Op(B) < 0, A – B tendrá el signo de A, que es positivo.

Se calcula Op(B):

$$Op(B) = NOT(B) + 1 = NOT(001011) + 1 = 110100 + 1 = 110101$$

Se suma:

$$A + Op(B)$$

							acarreo
	0	1	1	0	1	0	$A_{(Ca2}$
+	1	1	0	1	0	1	$Op(B)_{(Ca2}$
1	0	0	1	1	1	1	A – B

$$A - B = 011010_{(Ca2} - 001011_{(Ca2} = 001111_{(Ca2}$$

f) Producto por potencias de 2 en Ca2

El producto por potencias de 2 en Ca2 se basa en el desplazamiento del valor posicional despreciando los bits que excedan la longitud acotada.

EJEMPLO 35 : PRODUCTO POR POTENCIA DE 2
Se desea conocer el resultado de $000101_{(Ca2}$ por $16_{(10}$

$$000101_{(Ca2} \cdot 2^4 = 010000_{(Ca2}$$

→ Se han añadido 4 bits de valor 0 por la derecha y luego se ha suprimido todo bit que exceda la posición x_5 que acota la longitud de la secuencia inicial. Se verifica:

$$000101_{(Ca2} = 5_{(10}$$
$$2^4 = 16_{(10}$$

$$000101_{(Ca2} \cdot 2^4 = 5 \cdot 16 = 80_{(10} = 1010000 \rightarrow en\ 6\ bits \rightarrow 010000_{(Ca2}$$

EJEMPLO 36 : PRODUCTO POR POTENCIA DE 2

Se desea conocer el resultado de $11111_{(Ca2}$ por $8_{(10}$

$$11111_{(Ca2} \cdot 2^3 = 11000_{(Ca2}$$

→ Se han añadido 3 bits de valor 0 por la derecha y luego se ha suprimido todo bit que exceda la posición x_4 que acota la longitud de la secuencia inicial.

g) Extensión de bits en Ca2

La extensión de bits de una secuencia expresada en complemento a dos no se hace mediante la incorporación de ceros por la izquierda, sino mediante la incorporación de tantos unos (1) como corresponda.

O sea, la incorporación de 1 por la izquierda no modifica el valor de una secuencia en Complemento a 2.

EJEMPLO 37 : EXTENSIÓN DE BITS EN Ca2
Se desea conocer qué secuencia de 8 bits codificada en Ca2 es equivalente a $10100_{(Ca2}$

$$10100_{(Ca2} = -2^4 + 2^2 = -16 + 4 = -12_{(10}$$

Extendiendo en 3 bits la longitud de la secuencia, se alcanza el mismo resultado:

$$11110100_{(Ca2} = -2^4 + 2^2 = -16 + 4 = -12_{(10}$$

15.3. Producto y cociente por potencias de la base

Para un sistema posicional de base b, hoy se puede ver el PRODUCTO de cualquier cantidad por una potencia ENTERA de esa base b^k como:

- Un desplazamiento de la coma fraccionaria k posiciones hacia la DERECHA (la cantidad resultante es más GRANDE que la original) si el exponente es POSITIVO.

- Un desplazamiento de la coma fraccionaria k posiciones hacia la IZQUIERDA (la cantidad resultante es más PEQUEÑA que la original) si el exponente es NEGATIVO.

EJEMPLO 38 : PRODUCTO Y COCIENTE POR POTENCIAS DE LA BASE

$$AB49{,}07DE_{(16} \cdot 16^3_{(10} = AB4907D,E_{(16}$$

$$AB49{,}07DE_{(16} \cdot 16^{-2}_{(10} = AB,4907DE_{(16}$$

Para el COCIENTE de cualquier cantidad por una potencia ENTERA de esa base b^k como:

- Un desplazamiento de la coma fraccionaria k posiciones hacia la IZQUIERDA (la cantidad resultante es más PEQUEÑA que la original) si el exponente es POSITIVO.

- Un desplazamiento de la coma fraccionaria k posiciones hacia la DERECHA (la cantidad resultante es más GRANDE que la original) si el exponente es NEGATIVO.

$$AB49{,}07DE_{(16} : 16^{3}_{(10} = A, B4907DE_{(16}$$
$$AB49{,}07DE_{(16} : 16^{-2}_{(10} = AB4907, DE_{(16}$$

***Producto y cociente por potencias de base 2 en formato SM de coma fija

Lo anterior es totalmente válido en el sistema de representación binaria para números fraccionarios en SM de coma fija.

16. NÚMEROS FRACCIONARIOS

Los números fraccionarios presentan una parte fraccionaria no nula.
pueden incorporar signo.
hay dos formatos principales de representación:

- formato de coma fija
- formato de coma flotante

16.1. Representación en coma fija

Se basa en reservar una cantidad de bits de la secuencia de longitud n para la parte entera p y una cantidad de bits para la parte fraccionaria (o "mantisa") m de modo que:

$$n = p + m$$

a) Magnitud en coma fija: codificación y descodificación

La magnitud del número representado por una secuencia codificada en un formato de coma fija se calcula aplicando TFN a cada uno de los dos fragmentos de la secuencia: el que codifica la parte entera y el que codifica la parte fraccionaria. este proceso se puede ver como una descodificación de una secuencia binaria.

El proceso complementario, la codificación de un valor expresado respecto de una base dada En forma de secuencia de bits de coma fija se realiza mediante la aplicación del teorema del resto tanto a la parte entera del número dado como a la parte fraccionaria del número dado.

Puesto que el proceso de codificación está acotado a un número de bits, es común que se cometa un error que desvía la secuencia codificada respecto a la magnitud original que se desea representar.

Una vez encontrada la secuencia binaria que codifica el valor original, el error se gestiona mediante 2 técnicas de aproximación:

- Truncamiento se basa en la omisión de bits más allá de la longitud fijada

- Redondeo se ejecuta en 3 pasos:
  - Calcular la precisión que ofrece la longitud fraccionaria fijada
  - Sumar a la secuencia codificada la mitad de la precisión
  - Omitir el resto de las cifras fraccionarias más allá de la longitud fijada

EJEMPLO 39 : CODIFICACIÓN EN COMA FIJA CON REDONDEO

Se desea representar en base 2 el número fraccionario $27{,}35_{(10}$ en un formato de coma fija con 8 bits, 3 de los cuales son fraccionarios, utilizando el método de aproximación de redondeo y calcular el error cometido en la codificación.

Como la base de origen es 10 y la base de destino es distinta de 10, se aplica teorema del resto.

La parte entera se lee en orden inverso al de aparición, mientras que la fraccionaria en orden directo de aparición.

→ Para la parte entera, se puede tomar un atajo:

$$27_{(10} = 16 + 8 + 2 + 1 = 2^4 + 2^3 + 2^1 + 2^0 = 11011_{(2}$$

→ Para la parte fraccionaria:
De mayor a menor peso, o sea, de izquierda a derecha en el sentido de la flecha:

$$\begin{aligned} 0{,}35 \cdot 2 &= 0{,}7 = 0 + 0{,}7 \\ 0{,}7 \cdot 2 &= 1{,}4 = 1 + 0{,}4 \\ 0{,}4 \cdot 2 &= 0{,}8 = 0 + 0{,}8 \\ 0{,}8 \cdot 2 &= 1{,}6 = 1 + 0{,}6 \\ 0{,}6 \cdot 2 &= 1{,}2 = 1 + 0{,}2 \\ 0{,}2 \cdot 2 &= 0{,}4 = 0 + 0{,}4 \end{aligned}$$

Se identifica la periodicidad y se detiene el proceso.

La parte fraccionaria es: $0{,}0\widehat{10110}$

$$27,35_{(10} = 11011{,}0\widehat{10110}_{(2}$$

La parte entera ocupa 5 bits, así que coincide con el formato especificado.
Pero solo se dispone de 3 bits para la parte fraccionaria, que ocupa 6.
Para ajustarse al formato especificado se aproxima por REDONDEO.

Para ello:

1 Se identifica la precisión del formato dado: 3 bits → precisión $= 0{,}001$

2 Se calcula la mitad de la precisión: $0,001_{(2} : 2_{(10} = 0,0001_{(2}$

3 Se suma la mitad de la precisión a la cantidad a redondear:

$$11011,010110_{(2} + 0,0001_{(2} = 11011,011010_{(2}$$

4 Se trunca el exceso de cifras fraccionarias hasta tener solo 3:

$$11011{,}0\widehat{10110}_{(2} \approx 11011{,}011_{(2}$$

En el formato especificado, las cifras fraccionarias son implícitas: $27{,}35_{(10} \approx 11011011$

El error cometido se calcula como el valor absoluto de la diferencia entre la cantidad a representar X y la cantidad representada en el formato dado $\hat{X}$ (una estimación de X):

$$\varepsilon = |X - \hat{X}|$$

Para determinarlo, primero se evalúa la cantidad representada $\hat{X}$ en base 10:

$$\hat{X} = 11011,011_{(2} = 2^4 + 2^3 + 2^1 + 2^0 + \frac{1}{4} + \frac{1}{8} = 27,375_{(10}$$

Ahora basta con encontrar la diferencia entre ambas cantidades, expresada en base 10:

$$\varepsilon = |X - \hat{X}| = \left|27,35_{(10} - 27,375_{(10}\right| = 0,025_{(10}$$

EJEMPLO 40 : DESCODIFICACIÓN EN COMA FIJA

Se considera la secuencia de bits 10110110 que representa un número real en un formato de coma fija, con cuatro bits para la parte entera y cuatro bits para la parte fraccionaria. Se desea conocer qué valor en base 10 codifica la secuencia dada.

De acuerdo con el formato especificado, se separa en dos partes la secuenciada dada:

- Parte entera: $1011_{(2} = 8 + 2 + 1 = 11_{(10}$
- Parte fraccionaria: $0110_{(2} = 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} = \frac{1}{4} + \frac{1}{8} = 0,375$

Y la secuencia codifica el valor $11,375_{(10}$

b) Rango en coma fija

El rango de representación para números fraccionarios sin signo expresados en formato de coma fija es:

$$[0,2^{n-m} - 2^{-m}\,]$$

Donde:
n es la longitud total de la secuencia
m es el número de bits reservados para la parte fraccionaria

El rango de representación para números fraccionarios con signo expresados en formato de coma fija es:

$$[-2^{n-m} + 2^{-m}\,, 2^{n-m-1} - 2^{-m}\,]$$

Donde:
n es la longitud total de la secuencia
m es el número de bits reservados para la parte fraccionaria
el formato de representación en coma fija es compatible con el formato en signo de magnitud.
En caso de estar codificada en signo y magnitud, el bit más significativo de la secuencia corresponderá con el bit de signo (S=0 si la magnitud es positiva y S=1 si es negativa).

c) Precisión en coma fija

Una secuencia representada en coma fija con m bits fraccionarios tiene una precisión de 2^{-m}

EJEMPLO 41 : CÁLCULO DE PRECISIÓN DE FORMATO DE COMA FIJA
Se desea conocer la precisión de una secuencia X codificada en un formato de representación en coma fija de signo y magnitud con 3 bits reservados para la parte entera. La secuencia es:

$$X = 11001011_{(SM}$$

Se sabe que el primer bit es de signo y que los 3 siguientes corresponden a la parte entera, por tanto:

$$\underbrace{1}_{S}\ \underbrace{100}_{p}\ \underbrace{1011}_{m}{}_{(SM}$$

La mantisa fraccionaria es 1011, de longitud 4, por tanto, la precisión de este formato es de $2^{-4} = \frac{1}{16_{(10}}$

d) Suma y resta en coma fija (igual que SM)

La suma y la resta en el formato de coma fija en signo y magnitud se realiza de igual modo que para los números enteros en signo de magnitud:

- Sumar las magnitudes si son del mismo signo y manteniendo en el resultado el signo de los operandos
- Restar las magnitudes si son de signo opuesto y tomando el signo del operando con mayor magnitud.

e) Producto y cociente por potencia de base en coma fija: DESPLAZAMIENTO

El formato de representación de SM en coma fija establece una cantidad de bits de la secuencia reservados para la parte entera y otra cantidad de bits reservados para la parte fraccionaria, así como un bit de signo en la posición de mayor peso.

Toda operación que conlleva el desplazamiento de la coma fraccionaria hacia la derecha es susceptible de añadir 0 por la derecha. Esto supone ningún inconveniente en ningún sistema de representación.

En cambio, toda operación que conlleva el desplazamiento de la coma fraccionar y hacia la izquierda es susceptible de suprimir bits significativos, que podrían quedar excluidos del resultado de la operación debido a la cantidad de cifras fraccionarias que el formato especifique. Esto puede acarrear errores de representación en las operaciones.

EJEMPLO 42 : PRODUCTO Y COCIENTE POR POTENCIA DE BASE EN COMA FIJA

Se considera un sistema de representación en SM de coma fija en 8 bits con 4 de ellos fraccionarios.
Se tiene la secuencia X = $0110{,}0101_{(SM8}$
Se desea calcular X : 2^3

Se opera |X| : 2^3 y se observa: $110{,}0101_{(2} : 2^3_{(10} = 0{,}1100101_{(2}$

Pero como el formato especificado tan solo admite 4 bits fraccionarios, el resultado se trunca:

$$110{,}0101 : 2^3 = 0{,}1100101 \rightarrow 0{,}1100_{(SM8}$$

f) Extensión del número de bits de magnitud en coma fija

Es relevante notar que la extensión de bits en signo y magnitud en un formato de coma fija se realiza solamente sobre el fragmento de la secuencia que codifica la magnitud, excluyendo el bit más significativo de la operación, puesto que se trata del bit de signo.

EJEMPLO 43 : EXTENSIÓN DE BITS N COMA FIJA

La secuencia X = 1100101 codifica un número en signo de magnitud en un formato de coma fija de 7 bits en que hay 3 bits reservados para la parte fraccionaria. Se desea conocer la secuencia que codifica el mismo número en el mismo formato, pero con 9 bits el lugar de 7.

Se tiene: $X = 1100101_{(SM}$
Con: $S = 1$ $|X| = 100101$
$parte\ entera = 100$
$parte\ fraccionaria = 101$

La extensión de bits INCORRECTA sería añadir directamente 2 bits de valor 0 por la izquierda:

$$X \neq \underbrace{00}_{extensión}\ 1100101_{(SM,}$$

La extensión de bits CORRECTA sería añadir directamente 2 bits de valor 0 por la izquierda DE LA MAGNITUD:

$$X = 1\ \underbrace{00}_{extención}\ 100101_{(SM,}$$

16.2. Representación en coma flotante

Este formato permite la representación de cantidades muy grandes o bien muy pequeñas de forma que ocupen el menor número de dígitos posibles (lo cual favorece un mejor uso de la memoria disponible).

El formato de coma flotante representa números en la forma:

$$\pm M \cdot b^{exponente}$$

Donde:

M	representa una cantidad fraccionaria llamada mantisa.
b	es la base de numeración empleada.
exponente	es un número entero que permite la codificación eficiente.

La mantisa M suele estar normalizada mediante un bit implícito que permite ganar una posición adicional (y así ahorrar más memoria todavía). Cuando la mantisa está normalizada con este bit implícito, debe entenderse el número representado de la siguiente forma:

$$\pm 1, M \cdot b^{exponente}$$

En los computadores, la base de numeración es 2.

Habitualmente, se suele representar el exponente con un EXCESO, para optimizar el uso de los bits disponibles. Este EXCESO, al igual que la presencia de bit implícito en la mantisa normalizada, no está codificado propiamente en la secuencia visible, sino que se especifica en las características del formato.

Esto provoca que el exponente no se pueda leer directamente del campo EXP con que se representa el formato, sino que HAY QUE RESTAR EL EXCESO.

La codificación de este formato se estructura mediante 3 campos:

- Signo (S) — Adopta el valor 1 para cantidades negativas.
  Adopta el valor 0 para cantidades positivas.
- EXP — Contiene la secuencia en binario que codifica exponente + exceso:
  $$EXP = exponente + exceso$$
  El exceso no está codificado en el campoEXP.
- Mantisa (M) — magnitud fraccionaria (normalizada con bit implícito o sin él)
  El video implícito no está codificado en el campo M.

Se suele especificar el formato como una tabla que refleja las posiciones de la secuencia reservadas para cada campo:

S	EXPONENTE			MANTISA		
x_n	x_{n-1}		x_j	x_{j-1}		x_0

Por tanto, se cumple:

$$\underset{\substack{\text{según}\\ x_n}}{\pm} \underbrace{x_{n-1}x_{n-2}\dots x_0}_{\substack{M \text{ o bien } 1{,}M\\ \text{con bit implícito}}} \cdot 2^{EXP-exceso}$$

***EJEMPLO CODIF

***EJEMPLO DESCODIF

17. Representación de información alfanumérica

Se entiende por información alfanumérica aquella que está constituida por caracteres.

Es decir, aunque emplee símbolos numéricos, no representa cantidades, sino símbolos de lo que entendemos como alfabeto y símbolos denominados especiales.

Para su representación, se ha establecido internacionalmente un convenio estandarizado denominado ASCII.

La codificación ASCII se basa en la asignación de una cadena de bits única y exclusiva a cada 1 de los caracteres alfanuméricos.

Existen 2 versiones de ASCII: con tan solo 18 símbolos y otra extendida de 8 bits, con lo cual, permite la codificación de hasta 256 símbolos distintos.

18. Codificación de señales analógicas

Se pueden utilizar señales analógicas, es decir, no digitales (no basadas únicamente en 0 y 1) para codificar información.
Para ello, se registra la variación continúa de parámetros eléctricos como la tensión o la corriente.

Para operar con señales analógicas, los computadores ordinarios requieren de mecanismos para su digitalización, de modo que se representen las señales analógicas como señales digitales.

La digitalización de una señal analógica es un proceso que generalmente se da en 3 etapas:

- El muestreo, en que se obtienen valores de la señal analógica a intervalos regulares.
- La cuantificación, que consiste en la asignación de valores de entre un conjunto finito a cada 1 de los valores que se han muestreado.
- La codificación binaria, que es la asignación de los valores cuantificados a 0 y 1, de modo que el resultado sea una secuencia binaria.

19. Representación en exceso a M

La representación en exceso a M es una estrategia para la optimización y de los recursos de cálculo y representación que se basa en un cambio de origen sobre un conjunto de números enteros dado.

Consiste en tomar como referencia el número entero más pequeño que se desea representar, que se identifica como M y define el EXCESO en la representación y restarlo a todos los números del conjunto a representar, de modo que el más pequeño se convierte en 0 y el resto en números naturales, cuya representación siempre es más asequible.

EJEMPLO 44 :
Se desea representar el conjunto de números enteros del intervalo [-60,12].

Como el número más pequeño a representar es -60, el exceso es M = -60.
Si a cada extremo del intervalo [60,12] se le resta M, se tiene que los nuevos extremos del intervalo representado en exceso a M = -60:

$$-60 - M = -60 - (-60) = 0$$
$$12 - (-60) = 72$$

y el intervalo representado en exceso a -60 es: $[0,72]$