UNIVERSIDAD DE LA RIOJA

Estructura de Computadores

Tema 3:

ARITMÉTICA DEL COMPUTADOR

Dr. Iván Luis Pérez Barrón
Grupo de Computación Científica
(GRUCACI)

Contenidos:

3.1. Unidad aritmético-lógica (ALU)

3.2. Representación en coma fija

3.3. Aritmética en coma fija

3.4. Representación en coma flotante

3.5. Aritmética en coma flotante

Contenidos:

3.1. Unidad aritmético-lógica (ALU)

3.2. Representación en coma fija

3.3. Aritmética en coma fija

3.4. Representación en coma flotante

3.5. Aritmética en coma flotante

3.1. Unidad aritmético-lógica (ALU)

- Circuito del procesador que realiza:
  - **Operaciones aritméticas**: suma, resta, opuesto, multiplicación, división,…
    - Objeto de este tema.
  - **Operaciones lógicas**: not, and, or, xor,…
    - Estudiadas en el tema anterior.

3.1. Unidad aritmético-lógica (ALU)

- ENTRADAS:
  - **Registros**: contienen los operandos.
  - **Unidad de control**: proporciona un conjunto de señales de control, que ofrecen una combinación distinta para cada operación posible.
- SALIDAS:
  - **Registros**: recogen los resultados.
  - **Registro de estado**: contiene un conjunto de indicadores o flags, que son bits que se activan para reflejar diversos detalles del resultado (carry, overflow, signo, resultado cero, paridad,…).

3.1. Unidad aritmético-lógica (ALU)

- Objetivo de este tema: "**Aritmética del computador**".
- Veremos 2 tipos de aritmética:
  - **Coma fija**: fundamentalmente números enteros.
  - **Coma flotante**: números con parte fraccionaria.
- En ambos casos estudiaremos:
  - **Representación**: forma de expresar los números con ceros y unos.
  - **Aritmética**: algoritmos para realizar las operaciones.

Contenidos:

3.1. Unidad aritmético-lógica (ALU)

3.2. Representación en coma fija

3.3. Aritmética en coma fija

3.4. Representación en coma flotante

3.5. Aritmética en coma flotante

Contenidos:

3.2. Representación en coma fija

- 3.2.1. Números enteros sin signo
- 3.2.2. Representación en signo-magnitud (S-M)
- 3.2.3. Representación en complemento a dos (C-2)
- 3.2.4. Conversión entre longitudes de bits diferentes
- 3.2.5. Números no enteros

Contenidos:

3.2. Representación en coma fija

3.2.1. Números enteros sin signo

3.2.2. Representación en signo-magnitud (S-M)

3.2.3. Representación en complemento a dos (C-2)

3.2.4. Conversión entre longitudes de bits diferentes

3.2.5. Números no enteros

3.2.1. Números enteros sin signo

- Los números binarios que hemos manejado hasta ahora han sido enteros sin signo. Por ejemplo:

$$12 \text{ d} \rightarrow 1100 \text{ b}$$

- Son los **uint** (unsigned integer) que utilizamos en los lenguajes de programación.

- Por ejemplo, con uint8 (8 bits) se pueden representar 2^8 = 256 números, desde el 0 hasta el 255:

$$0000\ 0000 \text{ b} \rightarrow \quad 0 \text{ d}$$

$$\dots$$

$$1111\ 1111 \text{ b} \rightarrow 255 \text{ d}$$

- **Valor** (*A*) de un entero sin signo de *n* bits:

$$a_{n-1}a_{n-2} \dots a_1 a_0 \quad \rightarrow \quad A = \sum_{i=0}^{n-1} 2^i a_i$$

3.2.1. Números enteros sin signo

- ¿Cómo representar, por ejemplo, −10.25 d?

$$-10.25\ \text{d} \rightarrow -1010.01\ \text{b}$$

- Problema: en un computador sólo disponemos de ceros y unos.
- Por tanto:
  - **No hay signo –** → es necesario algún modo de representación para números enteros con signo.
    - Solución: **representación en coma fija**.
  - **No hay punto** → necesitamos un sistema de representación para números con parte fraccionaria.
    - Solución: **representación en coma flotante**.

Contenidos:

3.2. Representación en coma fija

3.2.1. Números enteros sin signo

3.2.2. Representación en signo-magnitud (S-M)

3.2.3. Representación en complemento a dos (C-2)

3.2.4. Conversión entre longitudes de bits diferentes

3.2.5. Números no enteros

3.2.2. Representación en signo-magnitud (S-M)

BS	Magnitud

$0 \rightarrow +$
$1 \rightarrow -$

- Representación:
  - **BS**: bit de signo (BS=0 → +; BS=1 → −).
  - **Magnitud**: valor absoluto (como los enteros sin signo).

- Ejemplo para una longitud de 8 bits:

  +25 d → **0**001 1001 b
  −25 d → **1**001 1001 b

- **Valor** (*A*) de un número de *n* bits:

$$\boldsymbol{a_{n-1}} a_{n-2} \ldots a_1 a_0 \;\rightarrow\; A = \begin{cases} +\displaystyle\sum_{i=0}^{n-2} 2^i a_i & \text{si} \quad \boldsymbol{a_{n-1} = 0} \\ -\displaystyle\sum_{i=0}^{n-2} 2^i a_i & \text{si} \quad \boldsymbol{a_{n-1} = 1} \end{cases}$$

3.2.2. Representación en signo-magnitud (S-M)

BS	Magnitud

$0 \rightarrow +$
$1 \rightarrow -$

- Inconvenientes:
  - **Suma y resta**: es necesario tener en cuenta los signos de los números (en complemento a dos será directo).
  - **Cero**: Hay dos representaciones del cero:

$$+0\text{ d} \rightarrow \mathbf{0}000\ 0000\text{ b}$$
$$-0\text{ d} \rightarrow \mathbf{1}000\ 0000\text{ b}$$

- La representación en signo-magnitud (S-M) no se utiliza en los computadores. En su lugar, se emplea la representación en complemento a dos (C-2).

Contenidos:

3.2. Representación en coma fija

3.2.1. Números enteros sin signo

3.2.2. Representación en signo-magnitud (S-M)

3.2.3. Representación en complemento a dos (C-2)

3.2.4. Conversión entre longitudes de bits diferentes

3.2.5. Números no enteros

3.2.3. Representación en complemento a dos (C-2)

- Son los **int** (integer) que utilizamos en los lenguajes de programación.

- Igual que en S-M, el primer bit es el **bit de signo** (BS=0 → +; BS=1 → −):

BS	

0 → +
1 → −

- Pero además de ser bit de signo, también tiene un **peso** asociado: el opuesto al que correspondería a su posición. Por ejemplo, para 4 bits:

−8	+4	+2	+1

BS:
0 → +
1 → −

- Ejemplo para una longitud de 4 bits:

+5 d → **0**101 b (+4+1)
−5 d → **1**011 b (**−8**+2+1)

3.2.3. Representación en complemento a dos (C-2)

−8	+4	+2	+1

BS:

0 → +

1 → −

- **Valor** (*A*) de un número de *n* bits en C-2:

$$\boldsymbol{a_{n-1}} a_{n-2} \ldots a_1 a_0 \;\rightarrow\; A = \begin{cases} +\displaystyle\sum_{i=0}^{n-2} 2^i a_i & \text{si} \quad \boldsymbol{a_{n-1} = 0} \\ \boldsymbol{-2^{n-1}} + \displaystyle\sum_{i=0}^{n-2} 2^i a_i & \text{si} \quad \boldsymbol{a_{n-1} = 1} \end{cases}$$

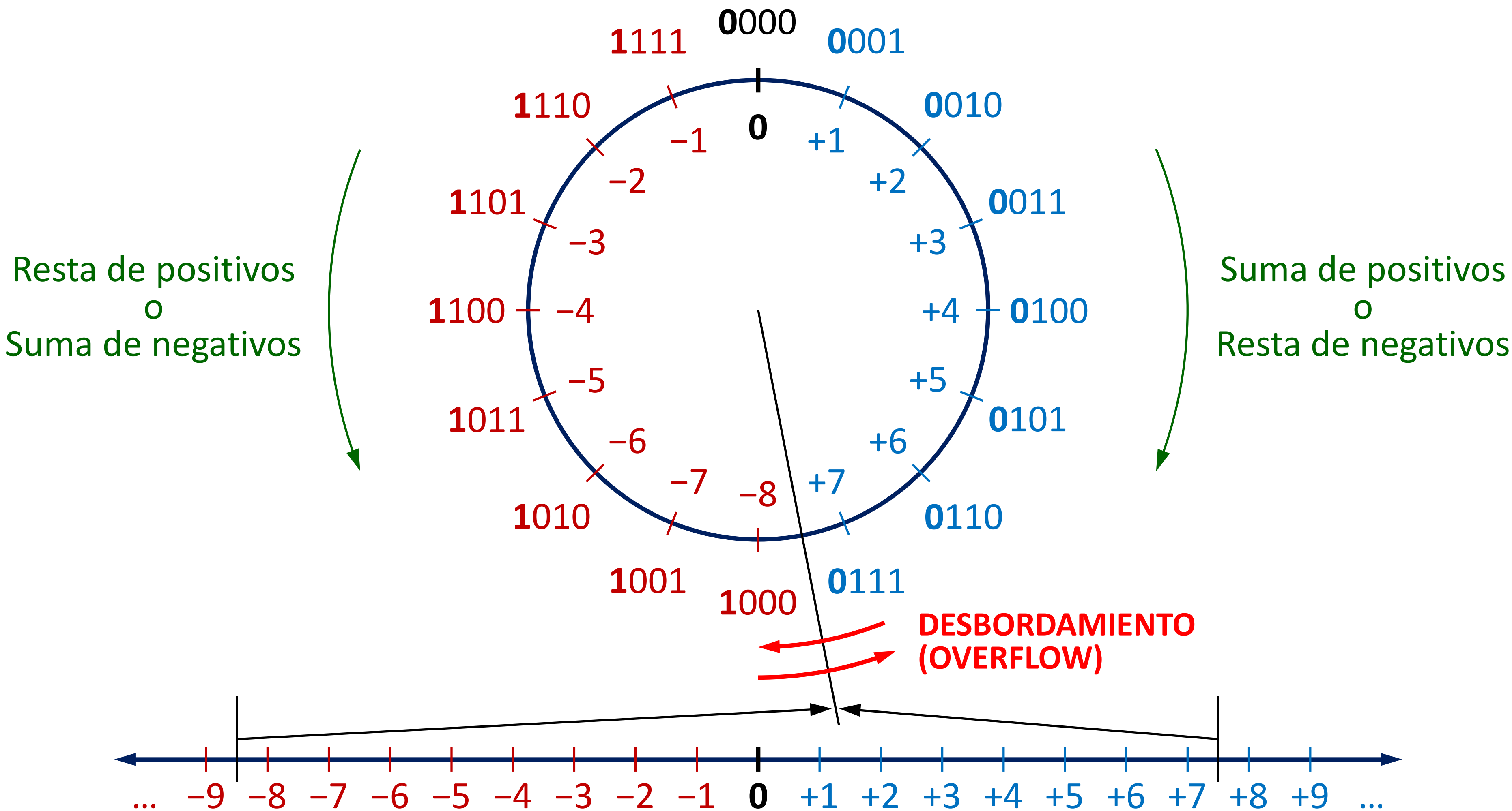
$$\rightarrow \; A = \boldsymbol{-2^{n-1} a_{n-1}} + \sum_{i=0}^{n-2} 2^i a_i$$

3.2.3. Representación en complemento a dos (C-2)

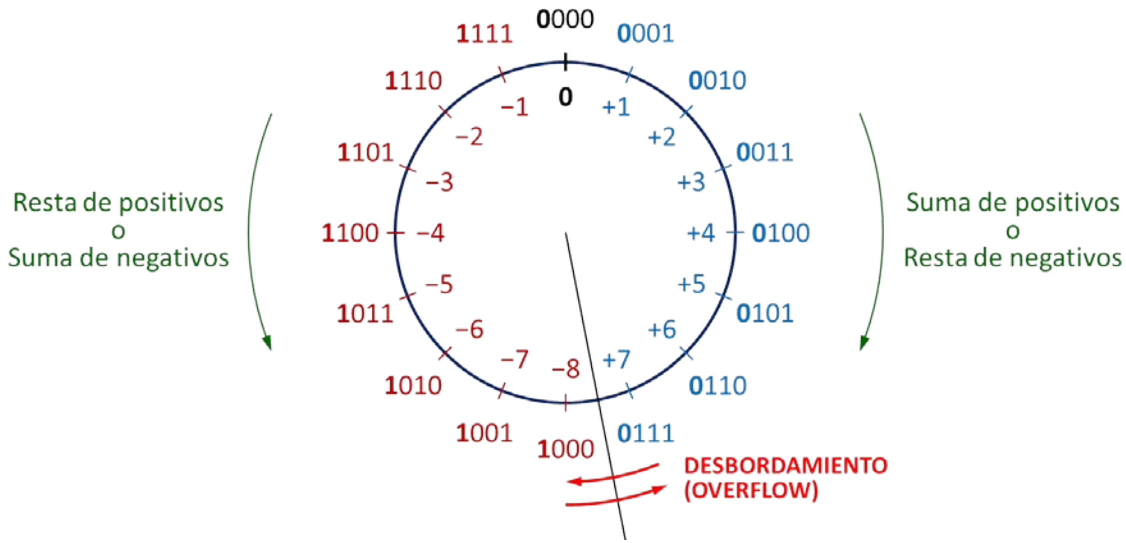
−8	+4	+2	+1

BS:
0 → +
1 → −

- **Rango** representable en C-2 con 4 bits: [−8,+7]
  - **Nº menor**: 1000 b → −8 d
  - **Nº mayor**: 0111 b → +7 d
- **Interpretación geométrica** de la representación en C-2:

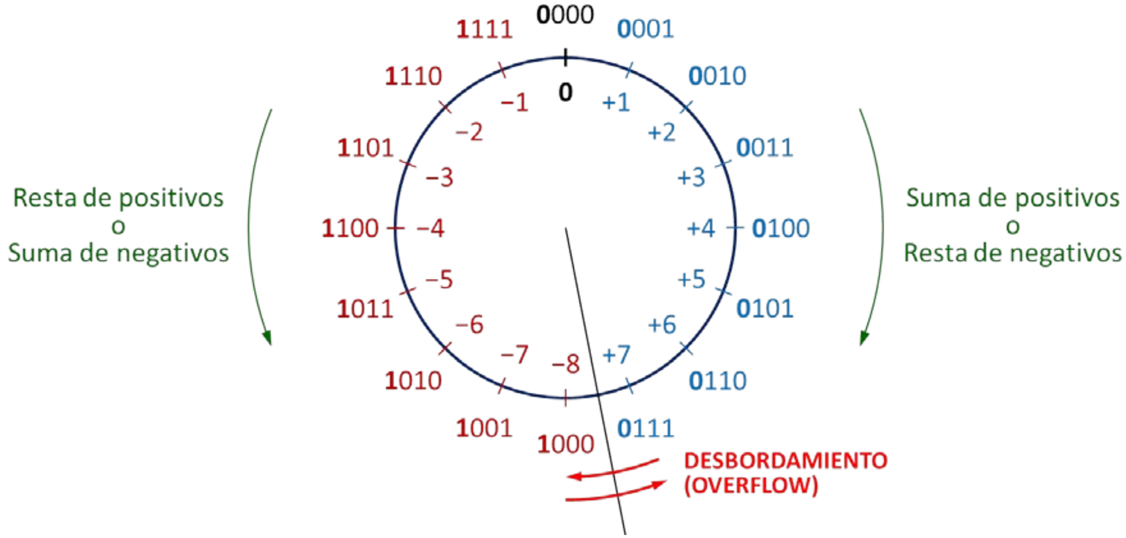


3.2.3. Representación en complemento a dos (C-2)



- **Sumar** (o restar un negativo): por ejemplo, partimos del +2 y avanzamos 3 posiciones en el sentido de las agujas del reloj, y llegamos al +5.
- **Restar** (o sumar un negativo): por ejemplo, partimos del +2 y avanzamos 3 posiciones en sentido contrario a las agujas del reloj, y llegamos al −1.
- **Desbordamiento (overflow)**: se produce cuando en una operación atravesamos el punto en el que se unieron los dos extremos del segmento tomado de la recta numérica.

3.2.3. Representación en complemento a dos (C-2)



- **Desbordamiento (overflow)**:
  - **(+) + (−)**: partiendo de la zona de los positivos y avanzando en sentido contrario a las agujas del reloj, o viceversa, es imposible llegar al desbordamiento:

$$1 + (-8) \rightarrow -7$$
$$(-1) + 7 \rightarrow +6$$

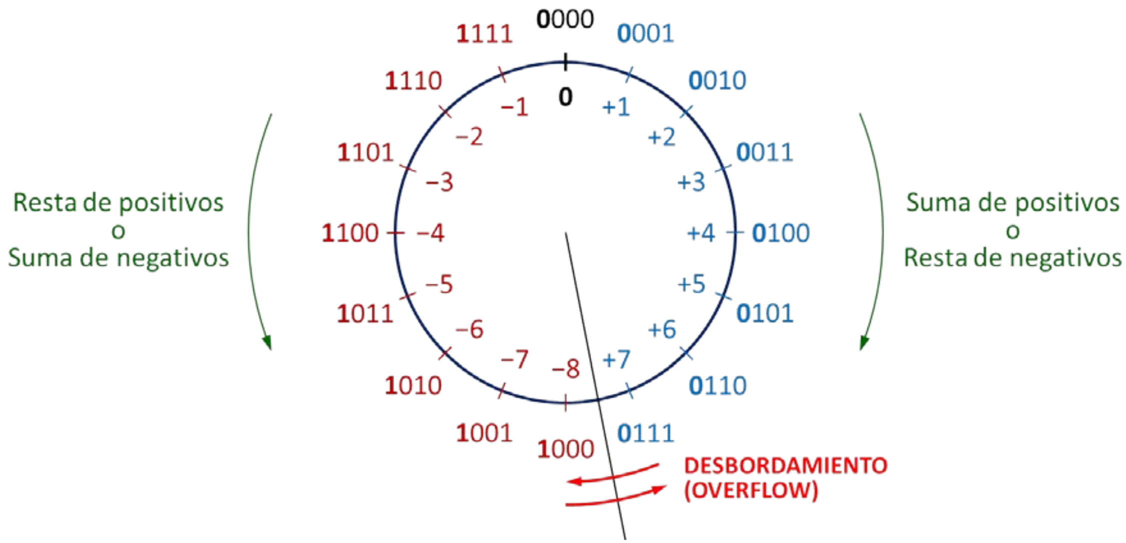
  - **(+) + (+)**: si se produce desbordamiento, se obtiene resultado (−):

$$5 + 4 \rightarrow -7$$

  - **(−) + (−)**: si se produce desbordamiento, se obtiene resultado (+):

$$-7 + (-3) \rightarrow +6$$

3.2.3. Representación en complemento a dos (C-2)



- **Regla del desbordamiento (overflow)**:
  - **BS operando 1 ≠ BS operando 2** → No overflow

    0001 + 1000 → No overflow

    1111 + 0111 → No overflow
  - **BS operando 1 = BS operando 2 ≠ BS resultado** → **Overflow**

    0101 + 0100 = 1001 → **Overflow**

    1001 + 1101 = 0110 → **Overflow**

- **Nota**: Más adelante veremos cómo se hace la suma en C-2.

3.2.3. Representación en complemento a dos (C-2)

- Comparación S-M vs C-2 (para 4 bits):

Decimal	S-M	C-2
+8	-	-
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
−0	1000	-
−1	1001	1111
−2	1010	1110
−3	1011	1101
−4	1100	1100
−5	1101	1011
−6	1110	1010
−7	1111	1001
−8	-	1000

3.2.3. Representación en complemento a dos (C-2)

- Comparación S-M vs C-2:
  - **Número de representaciones del cero**:
    - S-M: dos
    - C-2: una
  - **Rango de números representables** (para n (4) bits):
    - n (4) bits $\rightarrow$ 2^n (16) combinaciones
    - 2^n (16) números distintos $\rightarrow$ 2^{n-1} (8) negat. y 2^{n-1} (8) posit.
    - S-M:
      - Hay que incluir −0 entre los negat. y +0 entre los posit.
      - Negativos: $[-(2^{n-1} - 1), -0]$ ([−7, −0])
      - Positivos: $[+0, +2^{n-1} - 1]$ ([+0, +7])
      - **Rango**: $\mathbf{[-(2^{n-1} - 1), +2^{n-1} - 1]}$ (**[−7, +7]**)
    - C-2:
      - Sólo hay un cero (+0), que se incluye entre los posit.
      - Negativos: $[-2^{n-1}, -1]$ ([−8, −1])
      - Positivos: $[+0, +2^{n-1} - 1]$ ([+0, +7])
      - **Rango**: $\mathbf{[-2^{n-1}, +2^{n-1} - 1]}$ (**[−8, +7]**)

3.2.3. Representación en complemento a dos (C-2)

- Números negativos: conversión decimal ↔ C-2
  - **FORMA 1**: sumar los pesos de los **unos**.

−128	+64	+32	+16	+8	+4	+2	+1

BS:
0 → +
1 → −

1000 0100 b = −124 d (−128+4)
1001 0100 b = −108 d (−128+16+4)
1100 0000 b = −64 d (−128+64)
1110 0000 b = −32 d (−128+64+32)
1111 0000 b = −16 d (−128+64+32+16)

- Resulta sencillo deducir el rango de números representables:
  - Nº más negativo: 1000 0000 b = −128 d
  - Nº más positivo: 0111 1111 b = +127 d
    (64+32+16+8+4+2+1 → 128−1)
    [Propiedad suma de potencias de dos consecutivas]

3.2.3. Representación en complemento a dos (C-2)

- Números negativos: conversión decimal ↔ C-2
  - **FORMA 2**: sumar los pesos de los **ceros**.
    - Algoritmo: los pesos de los ceros tienen que sumar una unidad menos que el valor absoluto del nº negativo a representar:

      1110 b = −2 d (+1)
      1100 b = −4 d (+2+1 = +3)
      1011 b = −5 d (+4)
      1010 b = −6 d (+4+1 = +5)

    - Aplicación: método muy ágil cuando hay pocos ceros.
    - Justificación:

		←	+7	→	
	−8	+4	+2	+1	
−6 →	1	0	**1**	0	
Suma de pesos de los unos:	−8		**+2**		= −6 ← Quitar a 8 el peso del **uno** (2)
Suma de pesos de los ceros:		+4		+1	
		←	7**−2**	→	= +5 ← Quitar a 7 el peso del **uno** (2)

(En ambos casos hacemos lo mismo, pero con los ceros partimos de una unidad menos)

3.2.3. Representación en complemento a dos (C-2)

- Números negativos: conversión decimal ↔ C-2
  - **FORMA 3**: hacer el **opuesto** del correspondiente positivo.
    - Algoritmo: invertir todos los bits y sumar 1:

$$
\begin{array}{lr}
+3\text{ d} = 0011\text{ b} \rightarrow & 1100 \\
+ & \underline{\quad 1} \\
 & 1101\text{ b} = -3\text{ d} \\
+6\text{ d} = 0110\text{ b} \rightarrow & 1001 \\
+ & \underline{\quad 1} \\
 & 1010\text{ b} = -6\text{ d} \\
+4\text{ d} = 0100\text{ b} \rightarrow & 1011 \\
+ & \underline{\quad 1} \\
 & 1100\text{ b} = -4\text{ d}
\end{array}
$$

    - Procedimiento rápido: yendo de derecha a izquierda (←)…
      - Mantener todos los bits hasta el primer **1**, incluido.
      - Invertir el resto de bits.

Contenidos:

3.2. Representación en coma fija

3.2.1. Números enteros sin signo

3.2.2. Representación en signo-magnitud (S-M)

3.2.3. Representación en complemento a dos (C-2)

3.2.4. Conversión entre longitudes de bits diferentes

3.2.5. Números no enteros

3.2.4. Conversión entre longitudes de bits diferentes

- ¿Cómo extender la longitud en bits de un número?

- **S-M**: trasladar BS a la nueva posición y rellenar la magnitud con ceros.

BS	b_2	b_1	b_0	$\rightarrow$	**BS**	0	0	0	0	b_2	b_1	b_0

+5 d: 0101 b $\rightarrow$ 0000 0101 b

−5 d: 1101 b $\rightarrow$ 1000 0101 b

- **C-2**: rellenar con copias de BS (ceros en positivos, unos en negativos).

BS	b_2	b_1	b_0	$\rightarrow$	**BS**	BS	BS	BS	BS	b_2	b_1	b_0

+5 d: 0101 b $\rightarrow$ 0000 0101 b

−5 d: 1011 b $\rightarrow$ 1111 1011 b

  - Causa: hacer el opuesto implica invertir bits $\rightarrow$ los ceros de relleno de los positivos se convierten en unos de relleno en los negativos.

  - Recordar que todos los unos iniciales pueden reducirse a uno solo:

    1111 1101 b = 101 b = −3 d

Contenidos:

3.2. Representación en coma fija

- 3.2.1. Números enteros sin signo
- 3.2.2. Representación en signo-magnitud (S-M)
- 3.2.3. Representación en complemento a dos (C-2)
- 3.2.4. Conversión entre longitudes de bits diferentes
- **3.2.5. Números no enteros**

3.2.5. Números no enteros

- La representación en coma fija también permite expresar números no enteros, aunque **no es lo habitual**.

- Para ello, basta con situar la coma **coma en otra posición** distinta a la asignada por defecto, que es a la derecha del bit menos significativo.

- De este modo, se da cabida tanto a la parte entera como a la parte fraccionaria, y todos los valores quedan **escalados**. Por ejemplo, en C-2:

01 0100.01 b = +20.25 d
11 0101.01 b = −10.75 d

- Ahora bien, al ser una representación en coma fija, una vez establecida la nueva **posición de la coma debe mantenerse** durante toda la secuencia de cálculos.

- Ello obliga a tomar un **compromiso** entre el número de bits dedicados a la parte entera y a la parte fraccionaria.

- Esta falta de flexibilidad quedará resuelta con la **representación en coma flotante**.

Contenidos:

3.1. Unidad aritmético-lógica (ALU)

3.2. Representación en coma fija

3.3. Aritmética en coma fija

3.4. Representación en coma flotante

3.5. Aritmética en coma flotante

Contenidos:

3.3. Aritmética en coma fija

3.3.1. Opuesto

3.3.2. Suma y resta

3.3.3. Multiplicación

3.3.4. División

Contenidos:

3.3. Aritmética en coma fija

3.3.1. Opuesto

3.3.2. Suma y resta

3.3.3. Multiplicación

3.3.4. División

3.3.1. Opuesto

- **S-M**: invertir BS:

$$+3\text{ d} = 0011\text{ b} \leftrightarrow 1011\text{ b} = -3\text{ d}$$

- **C-2**: "*hacer el C-2 de n*" significa hacer el opuesto de *n* en C-2.
  - Algoritmo: invertir todos los bits y sumar 1:

$$+3\text{ d} = 0011\text{ b} \rightarrow \begin{array}{r} 1100 \\ +\underline{\quad 1} \\ 1101 \end{array} \text{ b} = -3\text{ d}$$

$$+6\text{ d} = 0110\text{ b} \rightarrow \begin{array}{r} 1001 \\ +\underline{\quad 1} \\ 1010 \end{array} \text{ b} = -6\text{ d}$$

$$+4\text{ d} = 0100\text{ b} \rightarrow \begin{array}{r} 1011 \\ +\underline{\quad 1} \\ 1100 \end{array} \text{ b} = -4\text{ d}$$

  - Procedimiento rápido: yendo de derecha a izquierda (←)…
    - Mantener todos los bits hasta el primer **1**, incluido.
    - Invertir el resto de bits.

3.3.1. Opuesto

- **C-2**:
  - Opuesto del opuesto: el número de partida:

$$
\begin{array}{rr}
+4\text{ d} = 0100\text{ b} \rightarrow & 1011 \\
+ & \underline{1} \\
 & 1100\text{ b} = -4\text{ d}
\end{array}
$$

$$
\begin{array}{rr}
-4\text{ d} = 1100\text{ b} \rightarrow & 0011 \\
+ & \underline{1} \\
 & 0100\text{ b} = +4\text{ d}
\end{array}
$$

  - Opuesto de cero: tiene que ser cero:

$$
\begin{array}{rr}
0\text{ d} = 0000\text{ b} \rightarrow & 1111 \\
+ & \underline{1} \\
 & \cancel{1}0000\text{ b} = 0\text{ d}
\end{array}
$$

    - Se genera un acarreo, pero en C-2 los acarreos se ignoran.

- **C-2**:
  - Opuesto de 10…0 b: caso a evitar:
    - Recordemos que, en C-2, el rango de números representables con n (4) bits es **asimétrico**; hay un negativo más que positivos: $[-2^{n-1}, +2^{n-1} - 1]$ ([−8, +7]):

$$10\ldots0\ b \rightarrow -2^{n-1} \qquad (-8)$$
$$01\ldots1\ b \rightarrow +2^{n-1} - 1 \qquad (+7)$$

    - El opuesto del **número más negativo** (10…0 b) no es representable con n bits.
    - Si se aplica el algoritmo para hacer el opuesto sobre este número, se obtiene el mismo valor, por lo que **este caso debe ser evitado**:

$$\begin{array}{rr} -8\ d = 1000\ b \rightarrow & 0111 \\ + & \underline{\quad 1} \\ & 1000\ \ b = -8\ d \end{array}$$

Contenidos:

3.3. Aritmética en coma fija

- 3.3.1. Opuesto
- **3.3.2. Suma y resta**
- 3.3.3. Multiplicación
- 3.3.4. División

3.3.2. Suma y resta

- La gran ventaja de la representación en C-2 es que permite hacer la suma de números con signo aplicando el **mismo procedimiento** básico de suma de enteros sin signo.

- Cuando intervienen números negativos, la interpretación de una misma suma es distinta como **enteros sin signo** y en **C-2**, pero en ambos casos es correcta:

$$\begin{array}{rrr} 1011 & (11) & (-5) \\ +\ \underline{0010} & +\ \underline{(\ \ 2)} & +\ \underline{(+2)} \\ 1101 & (13) & (-3) \end{array}$$

- Sin embargo, hay que tener en cuenta **dos diferencias**:
  - Acarreo (C):
    - Enteros sin signo: forma parte del resultado.
    - C-2: se ignora.

$$\begin{array}{rrr} 1011 & (11) & (-5) \\ +\ \underline{0111} & +\ \underline{(\ \ 7)} & +\ \underline{(+7)} \\ \mathbf{1}0010 & (18) & (+2) \\ (\mathbf{C})\quad & & \end{array}$$

- (…) Sin embargo, hay que tener en cuenta **dos diferencias**:
  - Acarreo (C).
  - Desbordamiento (overflow):
    - Enteros sin signo: se ignora.
    - C-2: debe comprobarse, para desechar el resultado en caso de producirse.

0101	(5)	(**+**5)
+ **0**111	+ (7)	+ (**+**7)
1100	(12)	(**−**4) → **Overflow**

- Recordemos la regla para detectar el desbordamiento:
  - **BS operando 1 ≠ BS operando 2** → No overflow
  - **BS operando 1 = BS operando 2 ≠ BS result.** → **Overflow**
- Puede producirse desbordamiento con o sin acarreo:

1000	(8)	(**−**8)
+ **1**111	+ (15)	+ (**−**1)
10111	(23)	(**+**7) → **Overflow**
(**C**)		

- **Ejemplos** de sumas y sus interpretaciones en C-2:

  - BS1 ≠ BS2:

$$\begin{array}{rr} & 1001 \\ + & \underline{0101} \\ & 1110 \end{array} \qquad \begin{array}{rr} & (-7) \\ + & \underline{(+5)} \\ & (-2) \end{array} \qquad\qquad \begin{array}{rr} & 1100 \\ + & \underline{0100} \\ & \mathbf{1}0000 \\ (\mathbf{C}) & \end{array} \qquad \begin{array}{rr} & (-4) \\ + & \underline{(+4)} \\ & (\ \ 0) \end{array}$$

  - BS1 = BS2 = BS result.:

$$\begin{array}{rr} & 0011 \\ + & \underline{0100} \\ & 0111 \end{array} \qquad \begin{array}{rr} & (+3) \\ + & \underline{(+4)} \\ & (+7) \end{array} \qquad\qquad \begin{array}{rr} & 1100 \\ + & \underline{1111} \\ & \mathbf{1}1011 \\ (\mathbf{C}) & \end{array} \qquad \begin{array}{rr} & (-4) \\ + & \underline{(-1)} \\ & (-5) \end{array}$$

  - BS1 = BS2 ≠ BS result.:

$$\begin{array}{rr} & \mathbf{0}101 \\ + & \underline{\mathbf{0}100} \\ & \mathbf{1}001 \end{array} \qquad \begin{array}{rr} & (\mathbf{+}5) \\ + & \underline{(\mathbf{+}4)} \\ & (\mathbf{-}7) \end{array} \rightarrow \textbf{Overflow} \qquad\qquad \begin{array}{rr} & \mathbf{1}001 \\ + & \underline{\mathbf{1}010} \\ & \mathbf{10}011 \\ (\mathbf{C}) & \end{array} \qquad \begin{array}{rr} & (\mathbf{-}7) \\ + & \underline{(\mathbf{-}6)} \\ & (\mathbf{+}3) \end{array} \rightarrow \textbf{Overflow}$$

3.3.2. Suma y resta

- **Resta en C-2**: al minuendo se le suma el opuesto del sustraendo, aplicando todo lo expuesto para la suma.

- **Ejemplos** de restas y sus interpretaciones en C-2:

$$
\begin{array}{rrr}
 & 0010 & (+2) \\
- & \underline{0111} & -\ \underline{(+7)}
\end{array}
\quad\rightarrow\quad
\begin{array}{rrr}
 & 0010 & (+2) \\
+ & \underline{1001} & +\ \underline{(-7)} \\
 & 1011 & (-5)
\end{array}
$$

$$
\begin{array}{rrr}
 & 0101 & (+5) \\
- & \underline{0010} & -\ \underline{(+2)}
\end{array}
\quad\rightarrow\quad
\begin{array}{rrr}
 & 0101 & (+5) \\
+ & \underline{1110} & +\ \underline{(-2)} \\
 & \mathbf{1}0011 & (+3) \\
 & (\mathbf{C}) &
\end{array}
$$

$$
\begin{array}{rrr}
 & 1011 & (-5) \\
- & \underline{0010} & -\ \underline{(+2)}
\end{array}
\quad\rightarrow\quad
\begin{array}{rrr}
 & 1011 & (-5) \\
+ & \underline{1110} & +\ \underline{(-2)} \\
 & \mathbf{1}1001 & (-7) \\
 & (\mathbf{C}) &
\end{array}
$$

- (...) **Ejemplos** de restas y sus interpretaciones en C-2:

$$\begin{array}{rrcrr} & 0101 & (+5) & & & 0101 & (+5) \\ - & \underline{1110} & -\ \underline{(-2)} & \rightarrow & + & \underline{0010} & +\ \underline{(+2)} \\ & & & & & 0111 & (+7) \end{array}$$

$$\begin{array}{rrcrr} & 0111 & (+7) & & & \mathbf{0}111 & (\mathbf{+}7) \\ - & \underline{1001} & -\ \underline{(-7)} & \rightarrow & + & \underline{\mathbf{0}111} & +\ \underline{(\mathbf{+}7)} \\ & & & & & \mathbf{1}110 & (\mathbf{-}2) \rightarrow \textbf{Overflow} \end{array}$$

$$\begin{array}{rrcrr} & 1010 & (-6) & & & \mathbf{1}010 & (\mathbf{-}6) \\ - & \underline{0100} & -\ \underline{(+4)} & \rightarrow & + & \underline{\mathbf{1}100} & +\ \underline{(\mathbf{-}4)} \\ & & & & & \mathbf{10}110 & (\mathbf{+}6) \rightarrow \textbf{Overflow} \\ & & & & & (\mathbf{C}) & \end{array}$$