Sistemas de Numeración

Circuitos Digitales EC1723

Universidad Simón Bolívar
Departamento de Electrónica y Circuitos
Prof. Juan. Claudio Regidor

Sistemas de Numeración Posicional

- En un número $a_n a_{n-1} a_{n-2} \cdots a_2 a_1 a_0$, cada símbolo a_i tiene un valor que depende de su posición, según la expresión:

$$N = a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + a_{n-2} \cdot b^{n-2} + \cdots a_2 \cdot b^2 + a_1 \cdot b^1 + a_0 \cdot b^0$$

$$N = \sum_{i=0}^{n} a_i \cdot b^i$$

- N es el valor numérico representado por la cadena de símbolos a_i.
- b es llamada la ***base*** del sistema.

Sistemas de Numeración Posicional

- Algunos ejemplos:

1512 (*base* 10)

$$N = 1 \cdot 10^3 + 5 \cdot 10^2 + 1 \cdot 10^1 + 2 \cdot 10^0$$

1512 (*base* 6)

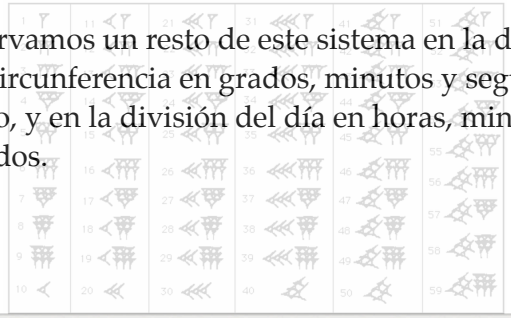
$$N = 1 \cdot 6^3 + 5 \cdot 6^2 + 1 \cdot 6^1 + 2 \cdot 6^0 = 404$$

1512 (*base* 8)

$$N = 1 \cdot 8^3 + 5 \cdot 8^2 + 1 \cdot 8^1 + 2 \cdot 8^0 = 842$$

Numeración Babilónica

- Los babilonios desarrollaron un sistema posicional de base 60, aunque los "dígitos" se formaban con sólo dos símbolos; no había un símbolo para el cero.
- Conservamos un resto de este sistema en la división de la circunferencia en grados, minutos y segundos de arco, y en la división del día en horas, minutos y segundos.



Numeración Maya

- Los mayas empleaban un sistema de numeración posicional de base 20.

$$N = a_4 \cdot 20^4 + a_3 \cdot 20^3 + a_2 \cdot 20^2 + a_1 \cdot 20^1 + a_0 \cdot 20^0$$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
																			

- Para las fechas ("cuenta larga"), el dígito a_1 solo llega hasta 17, de modo que las dos últimas cifras recorren los valores 0-359, aproximando la duración del año.

Numeración indo-arábiga

- El sistema decimal que usamos en la actualidad se deriva del desarrollado en India hacia el s. VI y adoptado por los árabes en el s. VIII. Leonardo de Pisa (Fibonacci) difundió el sistema en Europa en 1202 con su *Liber Abaci* (Libro del Ábaco).
- Los símbolos originales usados aún en los países musulmanes son diferentes de los empleados en Europa y luego en el resto del mundo.

٠	١	٢	٣	٤	٥	٦	٧	٨	٩
0	1	2	3	4	5	6	7	8	9

Sistemas de Numeración Posicional

- La base se denota como un subíndice del número, excepto cuando no hay ambigüedad.
- Nuestro sistema de base diez o decimal, por convención, no requiere el subíndice.
  - $100110101_2 = 465_8 = 309$
- Para sistemas cuya base sea mayor que diez, se suelen emplear letras como dígitos, según la equivalencia: A=10, B=11 … F=15 … K=20, etc.

Conversión de una base cualquiera a base 10

- Se evalúa la expresión de definición de sistema posicional.

$$465_8 = 4 \cdot 8^2 + 6 \cdot 8 + 5 = 309$$

$$2AF8_{16} = 2 \cdot 16^3 + 10 \cdot 16^2 + 15 \cdot 16 + 8 = 11000$$

$$21012_3 = 2 \cdot 3^4 + 1 \cdot 3^3 + 0 \cdot 3^2 + 1 \cdot 3 + 2 = 194$$

$$101010_2 = 1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2 = 42$$

Conversión de base 10 a una base cualquiera

$$\left(a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \cdots + a_2 \cdot b^2 + a_1 \cdot b^1 + a_0 \cdot b^0\right) \div b$$

$\Rightarrow$ Cociente: $a_n \cdot b^{n-1} + a_{n-1} \cdot b^{n-2} + \cdots + a_2 \cdot b^1 + a_1 \cdot b^0$

Residuo: a_0

$$\left(a_n \cdot b^{n-1} + a_{n-1} \cdot b^{n-2} + \cdots + a_2 \cdot b^1 + a_1 \cdot b^0\right) \div b$$

$\Rightarrow$ Cociente: $a_n \cdot b^{n-2} + a_{n-1} \cdot b^{n-3} + \cdots + a_2 \cdot b^0$

Residuo: a_1

- Cada división da como residuo a cada uno de los a_i, en orden del menos al más signficativo.

Conversión de base 10 a una base cualquiera

- Convertir 57 a base 4
  - 321_4

$3 \cdot 4^2 + 2 \cdot 4 + 1 = 57$

57	4		
1	14	4	
	2	3	4
		3	0

- Convertir 515 a base 8
  - $515 = 1003_8$

$1 \cdot 8^3 + 3 = 515$

515	8		
3	64	8	
	0	8	8
		0	1	8
			1	0

Conversión de una base a otra cualquiera

- Usamos el sistema decimal como paso intermedio.
- Ejemplo: Convertir 537_8 a base 5:
  - 537_8 a base 10: $5 \cdot 8^2 + 3 \cdot 8 + 7 = 351$
  - 351 a base 5:

351	5	
1	70	5
	0	14	5
		4	2

$\Rightarrow \quad 537_8 = 2401_5$

Caso particular: bases potencia de 2

- Nombres especiales:
  - Base 2: binario 10110100_2
  - Base 8: octal 264_8
  - Base 16: hexadecimal $B4_{16}$ ó $B4_H$
- Un dígito en el sistema octal es equivalente a 3 dígitos (*bits*) en el sistema binario.
- Un dígito en el sistema hexadecimal es equivalente a 4 bits.

Conversión entre bases potencia de 2

- Octal a binario: se sustituye cada dígito por los tres bits equivalentes, según la tabla:

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

- Hexadecimal a binario: se sustituye cada dígito por los cuatro bits equivalentes, según la tabla:

0	1	2	3	4	5	6	7
0000	0001	0010	0011	0100	0101	0110	0111

8	9	A	B	C	D	E	F
1000	1001	1010	1011	1100	1101	1110	1111

Conversión entre bases potencia de 2

- Ejemplos:
  - $3215_H = 0011\ 0010\ 0001\ 0101_2$
  - $8A4F_H = 1000\ 1010\ 0100\ 1111_2$
  - $32177_8 = 011\ 010\ 001\ 111\ 111_2$
  - $173102_8 = 001\ 111\ 011\ 001\ 000\ 010_2$
  - $132103_4 = 01\ 11\ 10\ 01\ 00\ 11_2$

Conversión entre bases potencia de 2

- Binario a octal: separamos los bits en grupos de 3, comenzando por el menos significativo, completando con ceros a la izquierda si es necesario. Cada grupo se sustituye entonces por su dígito octal equivalente.
- Binario a hexadecimal: separamos los bits en grupos de 4, comenzando por el menos significativo, completando con ceros a la izquierda si es necesario. Cada grupo se sustituye entonces por su dígito hexadecimal equivalente.

Conversión entre bases potencia de 2

- Ejemplos:
  - $100'1011'0100'1110'1111_2 = 4B4EF_H$
  - $1'1111'0000'0101'0010_2 = 1F052_H$
  - $11'010'110'111'100_2 = 32674_8$
  - $1'111'111'101'110'000_2 = 177560_8$
  - $1'10'11'00'00'11'10'01_2 = 12300321_4$

Conversión entre bases potencia de 2

- Podemos usar el sistema binario como paso intermedio para conversiones octal-hexadecimal:
  - $1A2B_H = 0001\ \ 1010\ \ 0010\ \ 1011_2$
  - $1A2B_H = 0'001'101'000'101'011_2 = 15053_8$
  - $173560_8 = 1\ \ 111\ \ 011\ \ 101\ \ 110\ \ 000_2$
  - $173560_8 = 1111'0111'0111'0000_2 = F770_H$
- Para convertir de decimal a binario, podemos hacer la conversión a octal o hexadecimal y luego a binario, a fin de reducir la cantidad de divisiones.

Conversión entre bases potencia de 2

- Convertir 12537 al sistema binario.
  - Usemos el sistema octal como paso intermedio

12537	8			
1	1567	8		
7	195	8		
3	24	8		
0	3			

  - $12537 = 30371_8 = 11\ 000\ 011\ 111\ 001_2$

Representación de números fraccionarios

- En el sistema decimal representamos la fracción 1572/100 como 15,72, es decir:

$$1 \cdot 10^1 + 5 \cdot 10^0 + 7 \cdot 10^{-1} + 2 \cdot 10^{-2}$$

- Para una base *b*, la representación posicional de un número fraccionario será

$$a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + ... + a_0 \cdot b^0 + a_{-1} \cdot b^{-1} + ... + a_{-m} \cdot b^{-m} + ...$$

$$N = \sum_{i=-m}^{n} a_i \cdot b^i$$

- La expresión anterior permite convertir un número en base *b* al sistema decimal.

Conversión de números fraccionarios

- $3122{,}21_4$ a base 10:

$$3 \cdot 4^3 + 1 \cdot 4^2 + 2 \cdot 4^1 + 2 \cdot 4^0 + 2 \cdot 4^{-1} + 1 \cdot 4^{-2} =$$

$$3 \cdot 64 + 16 + 2 \cdot 4 + 2 + 2 \cdot 0{,}25 + 0{,}0625 = 218{,}3125$$

- $4{,}4404_5$ a base 10:

$$4 \cdot 5^0 + 4 \cdot 5^{-1} + 4 \cdot 5^{-2} + 0 \cdot 5^{-3} + 4 \cdot 5^{-4} =$$

$$4 + 4 \cdot 0{,}2 + 4 \cdot 0{,}04 + 4 \cdot 0{,}0016 = 4{,}9664$$

Conversión de números fraccionarios

- Para convertir un número decimal a una base b cualquiera, la parte entera se trata igual que antes. Para la parte fraccionaria f observamos:

$$b \cdot f = b \cdot \left(a_{-1} \cdot b^{-1} + a_{-2} \cdot b^{-2} + a_{-3} \cdot b^{-3} + \ldots\right) = a_{-1} + a_{-2} \cdot b^{-1} + a_{-3} \cdot b^{-2} + \ldots$$

$$b \cdot \left(a_{-2} \cdot b^{-1} + a_{-2} \cdot b^{-2} + \ldots\right) = a_{-2} + a_{-3} \cdot b^{-1} + \ldots$$

- Esto es, al multiplicar la base por la parte fraccionaria obtenemos un número cuya parte entera es el valor del digito que sigue a la coma fraccionaria. Repitiendo el proceso con la nueva parte fraccionaria, obtenemos los dígitos sucesivos.

Conversión de números fraccionarios

- 0,22912 a base 5:
  - 0,22912 * 5 = 1,1456
  - 0,1456 * 5 = 0,728
  - 0,728 * 5 = 3,64
  - 0,64 * 5 = 3,2
  - 0,2 * 5 = 1,0 $\Rightarrow$ $0{,}10331_5$
- El proceso termina cuando la parte fraccionaria del producto es cero.

Conversión de números fraccionarios

- Un número fraccionario que tenga representación exacta en una base no necesariamente la tiene en otra base.
- Por ejemplo, la fracción $1/3 = 3^{-1}$ se representa como $0{,}1_3$, pero en el sistema decimal es la fracción periódica 0,33333…
- El proceso de conversión puede no terminar, produciendo así una fracción periódica.

Conversión de números fraccionarios

- Convertir 0,2 a base 2.
  - 0,2 * 2 = 0,4
  - 0,4 * 2 = 0,8
  - 0,8 * 2 = 1,6
  - 0,6 * 2 = 1,2
  - 0,2 * 2 = 0,4….
- A partir de aquí se repiten indefinidamente los mismos valores, y tenemos la fracción periódica binaria 0,00110011…..

Aritmética binaria

- Suma:

+	0	1
0	0	1
1	1	10

```
1 1 1 1   1          "acarreo" o carry
  1 0 1 1 0 1 0 +
  1 1 0 1 0 1 1
-------------------
1 1 0 0 0 1 0 1
```


Aritmética binaria

- Resta:

```
1 1 1 1   1          "pido" o borrow
1 1 0 0 0 1 0 1 –
  1 1 0 1 0 1 1
-------------------
  1 0 1 1 0 1 0
```

- ```
  1 0 1 0 1 0 1 +     1 1 0 1 1 1 0 1 +     1 1 0 0 0 1 0 1 –
  1 0 1 0 1 1 0       1 0 1 1 1 0 1         1 0 1 0 0 1 1
  -------------       -------------         -------------
  1 0 1 0 1 0 1 1     1 0 0 1 1 1 0 1 0       1 1 1 0 0 1 0
  ```


Representación de números negativos

- **Magnitud y signo.**
  - Es la forma que empleamos comúnmente:
    +157 –1249 –17,5732 255,998
  - En el sistema binario podemos usar un bit adicional para el signo. La convención normal es 0 para positivo y 1 para negativo:
    - +69 = 0 1000101 –69 = 1 1000101
    - –129 = 1 10000001

Representación de números negativos

- **Complemento a 1.**
  - La operación de complemento consiste en restar un número N de otro formado por tantos unos como bits tenga N. El resultado es el complemento a 1 de N:
    Comp. a 1 de 010010 111111 – 010010 = 101101
  - La forma práctica de hallar el complemento a 1 es cambiar el valor de cada bit, de 0 a 1 y viceversa. Así, el complemento a 1 de 100101110 es 011010001.
  ```

Representación de números negativos

- La representación en complemento a 1 de ***n*** bits usa el valor binario sin alterar para números positivos y el complemento a 1 para los números negativos, todos expresados en ***n*** bits.
  - +62 en C'1 de 8 bits: 00111110
  - –62 en C'1 de 8 bits: 11000001
- Se pueden expresar cantidades entre $-(2^{n-1}-1)$ y $2^{n-1}-1$. El cero tiene dos formas: 0 0...0 y 1 1...1.

Representación de números negativos

- Operaciones en complemento a uno:
  - Cambio de signo: se halla el complemento a uno del número.
- – (0 1001) = 1 0110; – (1 0101) = 0 1010
  - Resta: A – B = A + (complemento a uno de B) + 1

$$\begin{array}{r} 14 - \\ \underline{6} \\ 8 \end{array} \quad \Rightarrow \quad \begin{array}{r} 0\ 1110 - \\ 0\ 0110 \end{array} \qquad \begin{array}{r} 0\ 1110 + \\ \underline{1\ 1001} \\ 10\ 0111 + 1 = 0\ 1000 \end{array}$$

Representación de números negativos

- **Complemento a 2.**
  - La operación de complemento a dos en ***n*** bits consiste en restar un número N de 2^n.
  - El resultado es el complemento a 2 de N en n bits: 100000 – 010010 = 101110
  - La forma práctica de hallar el complemento a 2 es copiar cada bit empezando por el menos significativo, hasta llegar a un 1, que también se copia. Luego cambiar el valor de cada bit siguiente. Así, el complemento a 2 de 100101100 es 011010100.

Representación de números negativos

- La representación en complemento a 2 de ***n*** bits usa el valor binario sin alterar para números positivos y el complemento a 2 para los números negativos, todos expresados en ***n*** bits.
  - +62 en C'2 de 8 bits: 00111110
  - –62 en C'2 de 8 bits: 11000010
- Se pueden representar cantidades entre -2^{n-1} y $(2^{n-1})-1$. El cero tiene una sola forma: 0 0...0.

Representación de números negativos

- La expresión matemática para la representación en complemento a 2 de ***n*** bits es

$$N = -a_{n-1} \cdot 2^{n-1} + a_{n-1} \cdot 2^{n-2} + a_{n-2} \cdot 2^{n-3} + \cdots + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

$$N = -a_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i$$

- Notar que el bit de signo tiene peso negativo. Si es cero, no altera el valor del número binario; si es uno, entonces hace la operación de complemento a dos.

Representación de números negativos

- Operaciones en complemento a dos:
  - Cambio de signo: se halla el complemento a dos del número.
- – (0 1001) = 1 0111; – (1 0110) = 0 1010
  - Resta: A – B = A + (complemento a dos de B)

$$\begin{array}{r} 14 - \\ \underline{6} \\ 8 \end{array} \quad \Rightarrow \quad \begin{array}{r} 0\ 1110 - \\ 0\ 0110 \end{array} \qquad \begin{array}{r} 0\ 1110 + \\ \underline{1\ 1010} \\ 10\ 1000 \end{array}$$

Representación de números negativos

- El '1' sobrante que puede aparecer proviene de la operación de complementación:
- $A - B \Rightarrow A + (2^n - B) = 2^n + (A - B)$
- Las operaciones de suma y resta pueden resultar en la condición llamada **desborde** u *overflow* cuando el resultado no puede representarse en n bits.
  - El desborde puede ocurrir sólo cuando se suman números del mismo signo y se detecta porque el resultado tiene un signo distinto a los sumandos.

Representación de números negativos

- Todas las operaciones siguientes resultan en desborde:
- 102 + 48 = 150

$$\begin{array}{r} 01100110 + \\ 00110000 \\ \hline 10010110 \end{array}$$

- –46+(–85) =–131

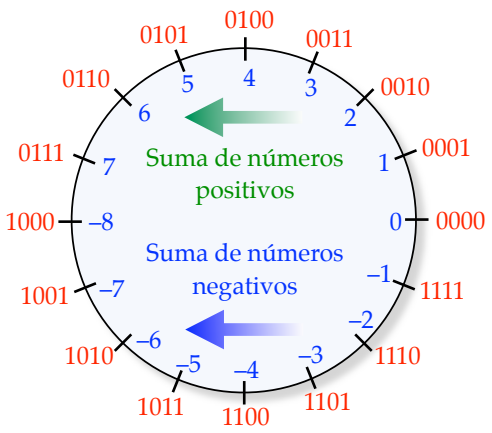
$$\begin{array}{r} 11010010+ \\ 10101011 \\ \hline 1\ 01111101 \end{array}$$

- 96 – (–60) = 96 + 60 = 156

01100000 – (11000100)

$$\begin{array}{r} 01100000 + \\ 00111100 \\ \hline 10011100 \end{array}$$

Representación de números negativos



Otros Códigos

- **BCD** (*Binary Coded Decimal* o Decimal codificado en binario).- Cada dígito de un número es representado por su equivalente binario de 4 bits. Se usa en aplicaciones de contabilidad, para evitar la imprecisión en las fracciones decimales representadas en binario.
  - 8,2 en binario es 1000,00110011…
  - 8,2 en BCD es 1000,0010, sin pérdida de precisión

Otros Códigos

- Hallar una suma en BCD requiere hacer una corrección en el resultado, debida al hecho de que hay 6 códigos que no representan ningún dígito.

```
                    1
  399+      0011 1001 1001+
  182       0001 1000 0010
  581       0101 0001 1011  ⇒  51B ??
                 110  110
            0101 1000 0001  ⇒  581
```


Otros Códigos

- **Código exceso-n**: Cada número es representado por su equivalente binario aumentado en n unidades.
- Ejemplo: en el código exceso-3, 0 se representa 0011, 1 es 0100, 4 es 0111, etc.
- En el código exceso-127 de 8 bits, los números del rango –127 a +128 se representan 00000000 a 11111111. Este código se usa para los exponentes en la representación de números en "punto flotante".

Otros Códigos

- **Código Gray**: a cada número se le asigna un valor binario tal que dos números consecutivos difieren solamente en un bit.
- El código de n bits se construye "reflejando" el de n–1 bits, según el procedimiento siguiente:

0	00	000	Gray	000 Binario
1	01	001		001
	11	011		010
	10	010		011
		110		100
		111		101
		101		110
		100		111

Otros Códigos

- **Códigos alfanuméricos:**
  - **EBCDIC** (*Extended Binary Coded Decimal Interchange Code*). Desarrollado por IBM como una extensión de BCD para representar letras y signos de puntuación.
  - **ASCII** (*American Standard Code for Information Interchange*). Es un código de 7 bits capaz de representar letras, números y signos de puntuación, y señales de control para un terminal.

Otros Códigos

- **Código Unicode**.- El código ASCII no permite representar los signos diacríticos usados en los idiomas europeos, ni otras formas de escritura. Para ello se creó Unicode, un código de 21 bits que acomoda gran variedad de alfabetos europeos o asiáticos (griego, cirílico, árabe, hebreo, etc.), ideogramas chinos, etc.
- Griego: Ελληνική Cirílico: Россия
- Árabe: العربية السعودية Hebreo: יִשְׂרָאֵל
- Katakana: ニポヌ Chino: 中华人民共和国

Otros Códigos

- Código Unicode
  - Plano multilingüe básico: 0000_H-$FFFF_H$
    - A = 0041_H ñ = $00F1_H$ α = $03B1_H$ ポ= $30DD_H$
  - Plano multilingüe suplementario: 10000_H-$1FFFF_H$
    - 𝄞 = $1D11E_H$ ∇ = $1D6C1_H$
  - Plano ideográfico suplementario: 20000_H-$2FFFF_H$
    - 龥 = $21CD2_H$ 𪾔 = $2F994_H$
  - Plano suplementario de propósitos especiales: $E0000_H$-$EFFFF_H$
  - Planos de uso privado: $F0000_H$-$FFFFF_H$ y 100000_H-$10FFFF_H$

Representación de Números en Punto Flotante

- Similar a la notación científica usual:
  - $2{,}125 \times 10^2 = 212{,}5 = 11010100{,}1_2 = (1{,}10101001 \times 10^{111})_2$
- Estándar IEEE 754-1985 / IEC 559
  - Define formatos de precisión simple, simple extendida, doble y doble extendida.
  - Incluye valores especiales: ∞, –∞, NaN (*Not a Number*)

Estándar IEEE 754-1985

- Formato de precisión simple

  - s: bit de signo (rep. de magnitud y signo)
  - Exponente binario en exceso 127
  - Mantisa normalizada, sin el bit más significativo
  - Números entre $2{,}35 \times 10^{-38}$ y $3{,}4 \times 10^{38}$ (7 cifr. sig.)
  - 0 10000110 10101001000000000000000 $\Rightarrow 1{,}10101001 \times 10^{111}$

Estándar IEEE 754-1985

- Formato de precisión doble

  - s: bit de signo (rep. de magnitud y signo)
  - Exponente binario en exceso 1023
  - Mantisa normalizada, sin el bit más significativo
  - Números entre $4{,}45 \times 10^{-308}$ y $1{,}8 \times 10^{308}$ (16 cifr. sig.)
  - 0 10000000110 1010100100000000...000 $\Rightarrow 1{,}10101001 \times 10^{111}$

Estándar IEEE 754-1985

- Valores especiales:
  - 0: exponente 0, mantisa 0
  - ±∞: exponente 255 ó 2047, mantisa = 0
  - NaN: exponente 255 ó 2047, mantisa ≠ 0
- Números desnormalizados:
  - exponente 0, mantisa = f ≠ 0, representa 0,f x 2^{emin}, con emin –126 ó –1022