Sistemas Numéricos, Aritmética Digital y Códigos

UCR – ECCI

CI-1204 Matemática Discretas

Prof. M.Sc. Kryscia Daviana Ramírez Benavides

Sistemas Numéricos

- Los sistemas de numeración son conjuntos de dígitos usados para representar cantidades, así se tienen los sistemas de numeración decimal, binario, octal, hexadecimal, romano, etc.
- Los cuatro primeros se caracterizan por tener una base (número de dígitos diferentes) mientras que el sistema romano no posee base y resulta más complicado su manejo tanto con números, así como en las operaciones básicas.

Sistemas Numéricos (cont.)

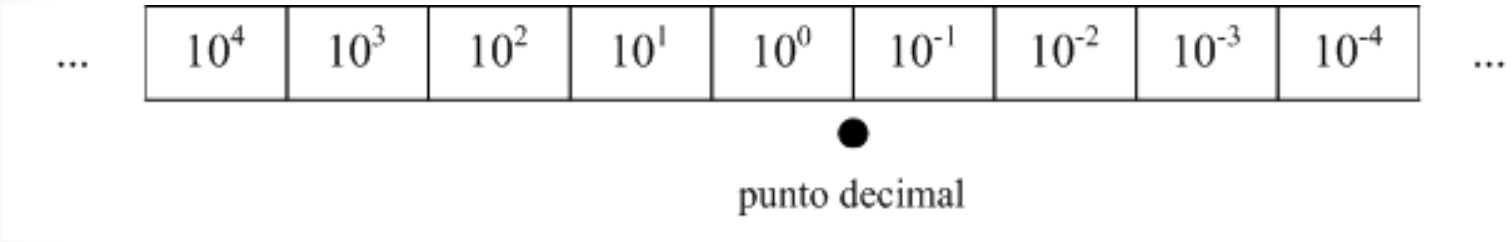
- Los sistemas de numeración que poseen una base tienen la característica de cumplir con la notación posicional, es decir, la posición de cada número le da un valor o peso.
- Así el primer dígito de derecha a izquierda después del punto decimal, tiene un valor igual a *b* veces el valor del dígito, y así el dígito tiene en la posición *n* un valor igual a $(b^n)*A$, donde:
  - *b* = valor de la base del sistema.
  - *n* = posición del dígito.
  - *A* = dígito.

Sistema Decimal o Base 10

- El sistema de numeración decimal es el más usado, tiene como base el número 10; o sea, que posee 10 dígitos (o símbolos) diferentes: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- El sistema de numeración decimal fue desarrollado por los hindúes, posteriormente lo introducen los árabes en Europa, donde recibe el nombre de sistema de numeración decimal o arábigo.
- Si se aplica la notación posicional al sistema de numeración decimal entonces el dígito en la posición *n* tiene el valor: $(10^n)*A$.

Sistema Decimal o Base 10 (cont.)

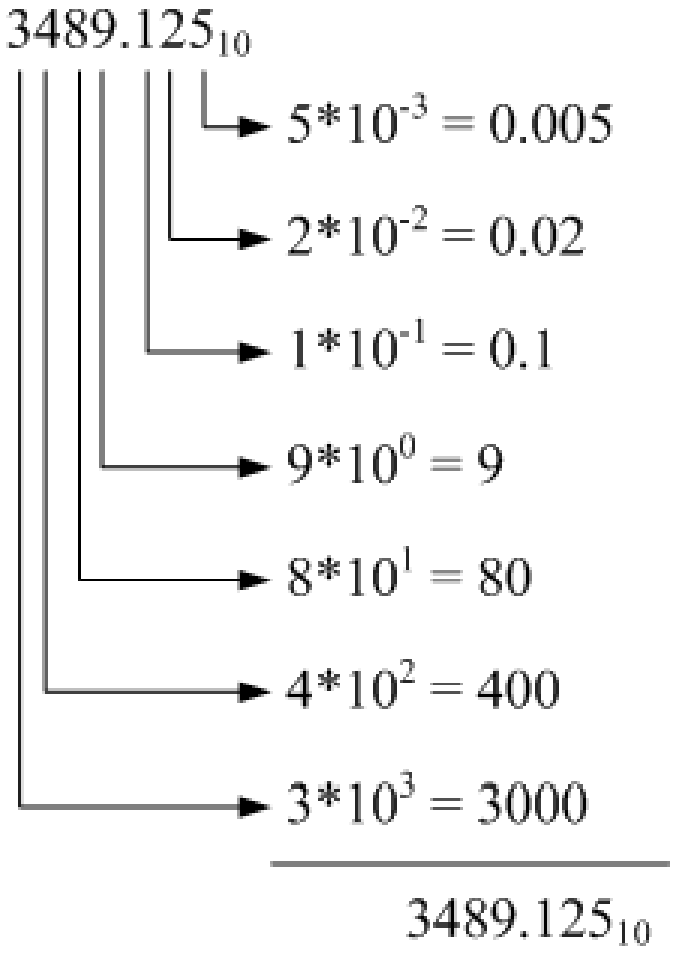
- Notación posicional del sistema:



...10000 1000 100 10 1 0.1 0.01 0.001 0.0001...

Sistema Decimal o Base 10 (cont.)

- Ejemplo:

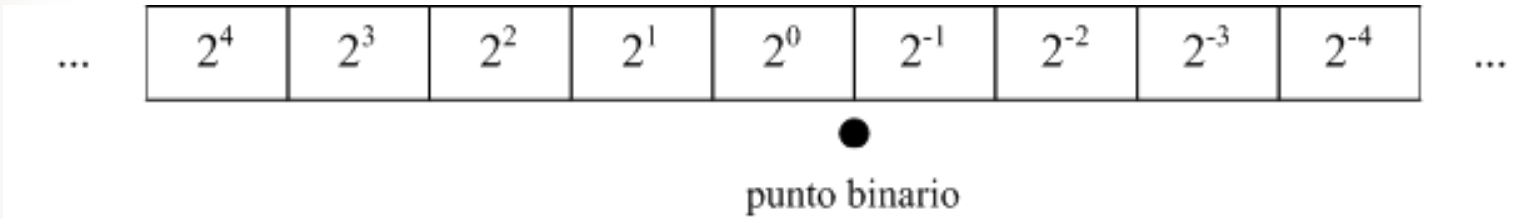


Sistema Binario o Base 2

- El sistema de numeración más simple que usa la notación posicional es el sistema de numeración binario; como su nombre lo indica, usa solamente dos dígitos (0,1).
- Si se aplica la notación posicional al sistema de numeración binario entonces el dígito en la posición *n* tiene el valor: $(2^n)*A$.
- Por simplicidad y poseer únicamente dos dígitos diferentes, este sistema se usa en computación para el manejo de datos e información.
  - A la representación de un dígito binario se le llama bit (de la contracción *binary digit*) y al conjunto de 8 bits se le llama byte. Ejemplo: 110 contiene 3 bits, 1001 contiene 4 y 1 contiene 1 bit.

Sistema Binario o Base 2 (cont.)

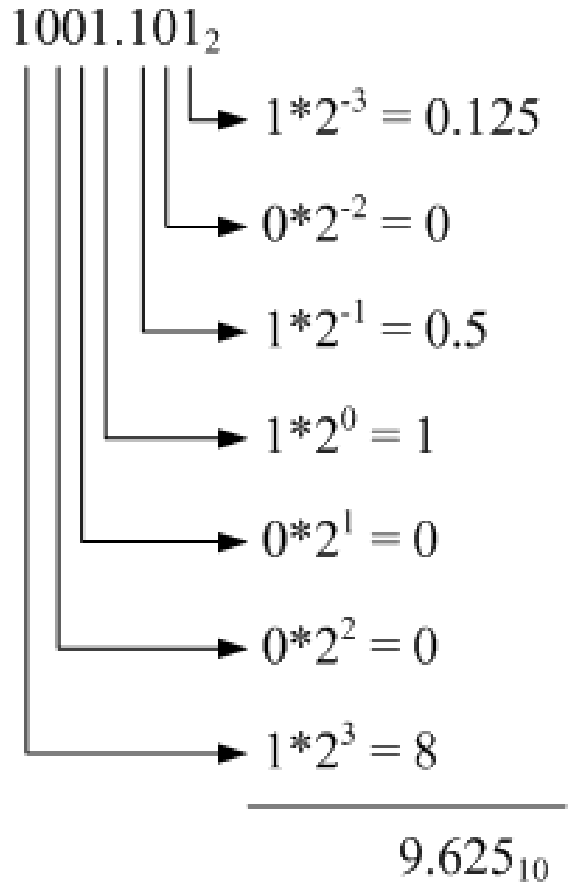
- Notación posicional del sistema:



...10000 1000 100 10 1 0.1 0.01 0.001 0.0001...

Sistema Binario o Base 2 (cont.)

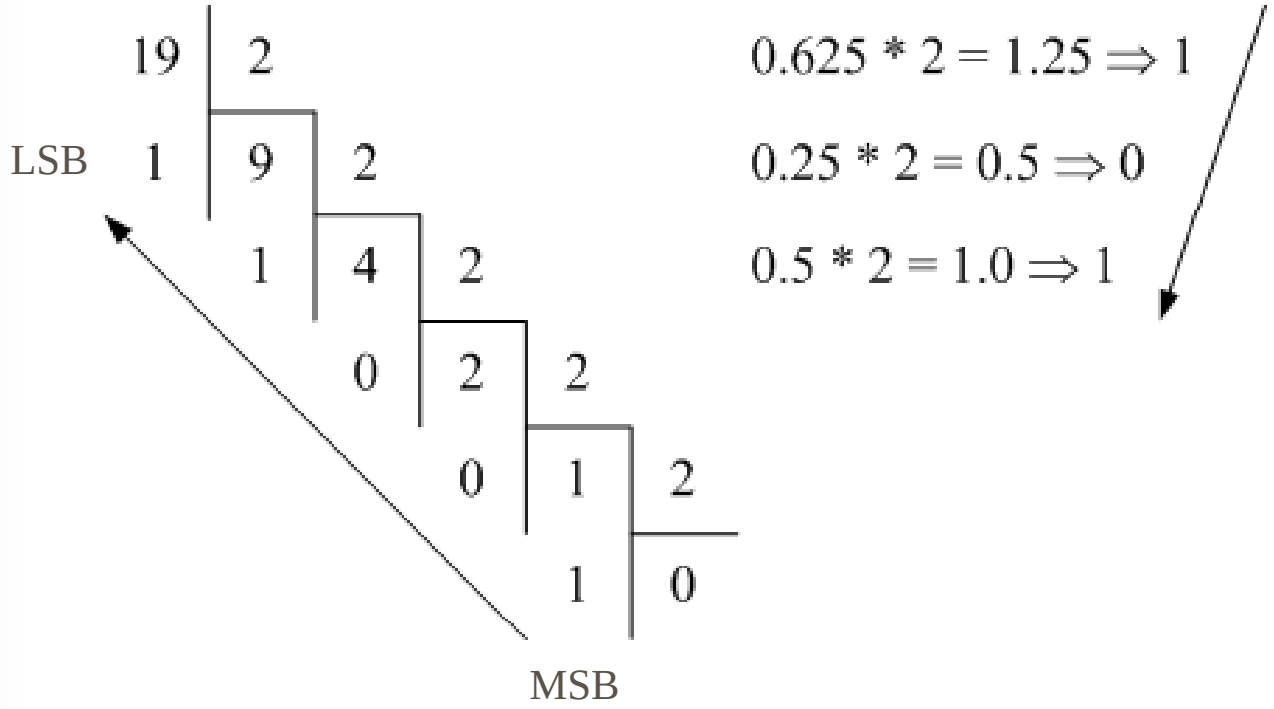
- Conversión de binario a decimal:



Sistema Binario o Base 2 (cont.)

- Conversión de decimal a binario:

$19.625_{10} = 10011.101_2$

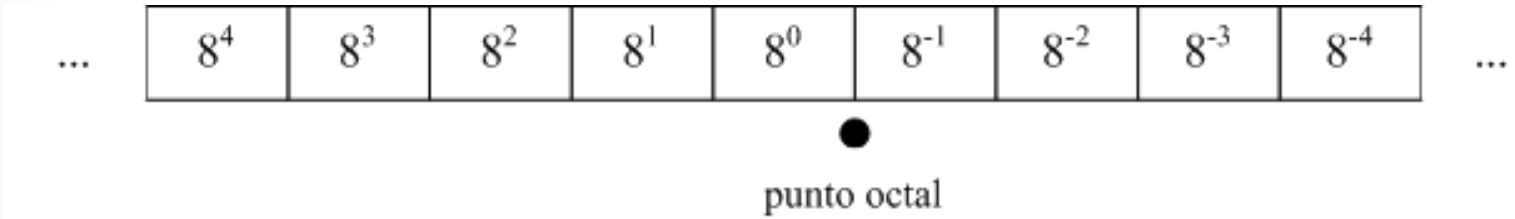


Sistema Octal o Base 8

- El sistema de numeración octal es también muy usado en la computación por tener una base que es potencia exacta de 2 o de la numeración binaria.
- Esta característica hace que la conversión a binario o viceversa sea bastante simple.
- El sistema octal usa 8 dígitos: 0,1,2,3,4,5,6,7; y tienen el mismo valor que en el sistema de numeración decimal.
- Si se aplica la notación posicional al sistema de numeración binario entonces el dígito en la posición *n* tiene el valor: $(8^n)*A$.

Sistema Octal o Base 8 (cont.)

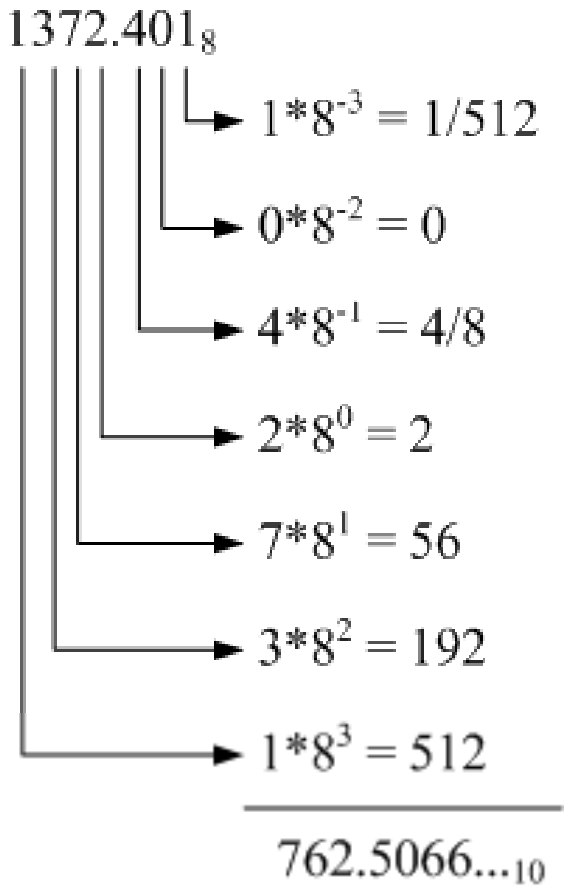
- Notación posicional del sistema:



...10000 1000 100 10 1 0.1 0.01 0.001 0.0001...

Sistema Octal o Base 8 (cont.)

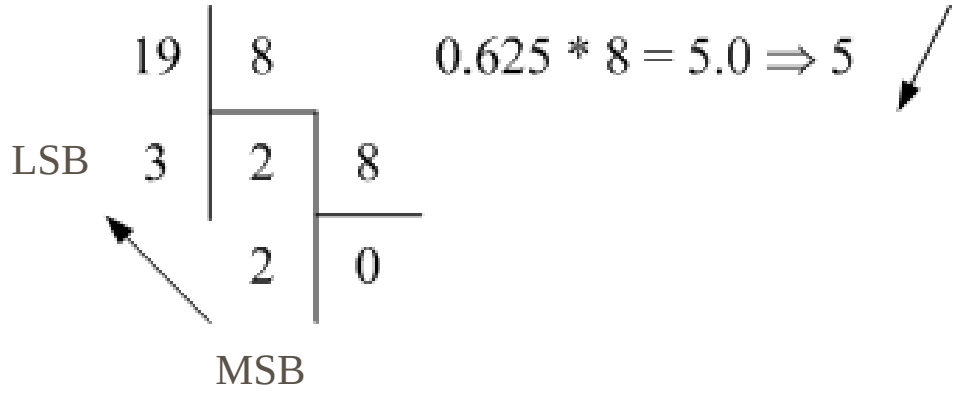
- Conversión de octal a decimal:



Sistema Octal o Base 8 (cont.)

- Conversión de decimal a octal:

$19.625_{10} = 23.5_8$



Sistema Octal o Base 8 (cont.)

- Conversión de binario a octal:

$$(1001).(01)_2 = 23.5_8$$

$010 = 2 \qquad 101 = 5$

$011 = 3$

- Conversión de octal a binario:

$$23.5_8 = 10011.101_2$$

$2 = 010 \qquad 5 = 101$

$3 = 011$

Sistema Hexadecimal o Base 16

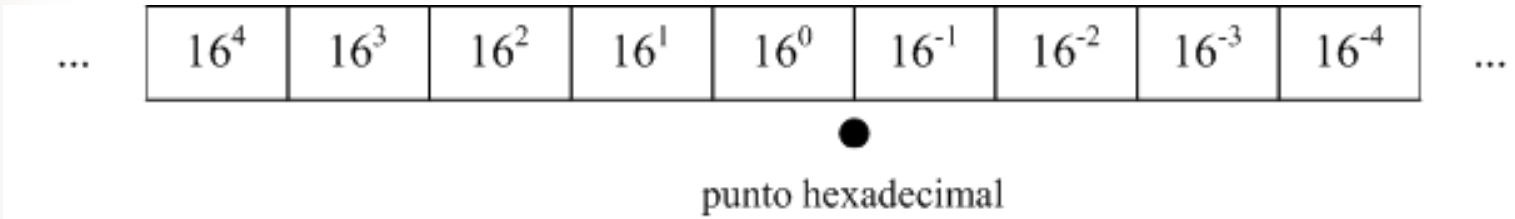
- Un gran problema con el sistema binario es la longitud para representar los números.
- Como las computadoras trabajan en sistema binario y aunque es posible hacer la conversión entre decimal y binario, ya vimos que no es precisamente una tarea cómoda.
- El sistema de numeración hexadecimal, o sea de base 16, resuelve este problema
- El sistema hexadecimal es compacto y nos proporciona un mecanismo sencillo de conversión hacia el formato binario, por lo que la mayoría del equipo de cómputo actual utiliza el sistema numérico hexadecimal.

Sistema Hexadecimal o Base 16 (cont.)

- El sistema hexadecimal usa 16 dígitos: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
- Si se aplica la notación posicional al sistema de numeración binario entonces el dígito en la posición *n* tiene el valor: $(16^n)*A$.

Sistema Hexadecimal o Base 16 (cont.)

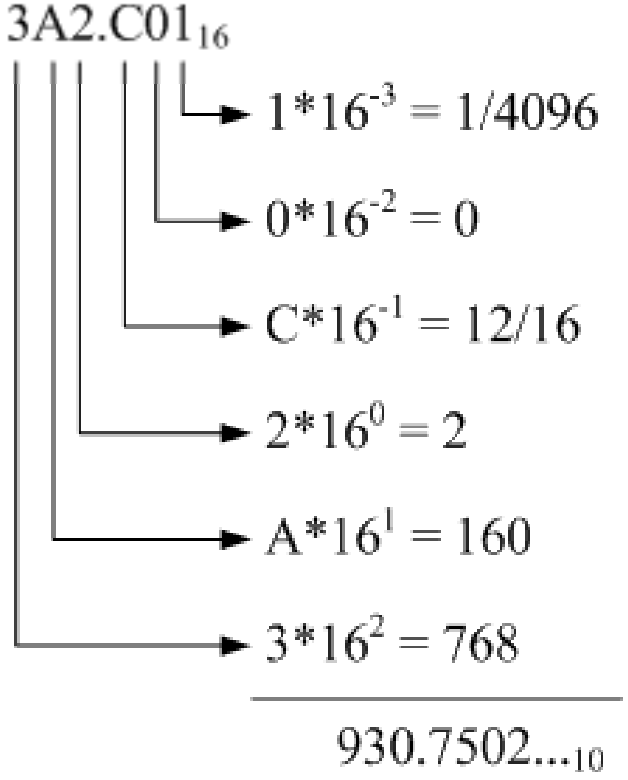
- Notación posicional del sistema:



...10000 1000 100 10 1 0.1 0.01 0.001 0.0001...

Sistema Hexadecimal o Base 16 (cont.)

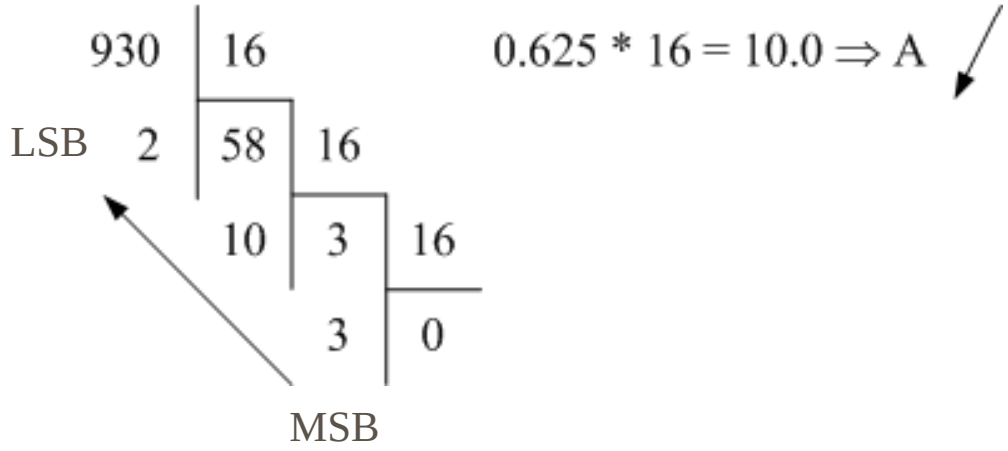
- Conversión de hexadecimal a decimal:



Sistema Hexadecimal o Base 16 (cont.)

- Conversión de decimal a hexadecimal:

$930.625_{10} = 3A2.A_{16}$



Sistema Hexadecimal o Base 16 (cont.)

- Conversión de binario a hexadecimal:

$$0001.101_2 = 13.A_{16}$$

$0001 = 1 \qquad 1010 = A$

$0011 = 3$

- Conversión de hexadecimal a binario:

$$13.A_{16} = 10011.101_2$$

$1 = 0001 \qquad A = 1010$

$3 = 0011$

Sistemas Numéricos

Base 10	Base 2	Base 8	Base 16
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Métodos de Conversión

Conversión	Método	Ejemplo
Binario a		
Octal	Sustitución	$110011001_2 = 110\ 011\ 001_2 = 631_8$
Hexadecimal	Sustitución	$110011001_2 = 1\ 1001\ 1001_2 = 199_{16}$
Decimal	Suma	$110011001_2 =$ $1*256+1*128+0*64+0*32+1*16+1*8+0*4+0*2+1*1= 409_{10}$
Octal a		
Binario	Sustitución	$1234_8 = 001\ 010\ 011\ 100_2 = 1010011100_2$
Hexadecimal	Sustitución	$1234_8 = 001\ 010\ 011\ 100_2 = 10\ 1001\ 1100_2 = 29C_{16}$
Decimal	Suma	$1234_8 = 1*512+2*64+3*8+4*1 = 668_{10}$

Métodos de Conversión (cont.)

Conversión	Método	Ejemplo
Hexadecimal a		
Binario	Sustitución	$C0E_{16} = 1100\ 0000\ 1110_2 = 110000001110_2$
Octal	Sustitución	$C0E_{16} = 1100\ 0000\ 1110_2 = 110\ 000\ 001\ 110_2 = 6016_8$
Decimal	Suma	$C0E_{16} = 12*256+0*16+14*1 = 3086_{10}$
Decimal a		
Binario	División-Multiplicación	$108_{10} = 1101100_2$
Octal	División-Multiplicación	$108_{10} = 154_8$
Hexadecimal	División-Multiplicación	$108_{10} = 6C_{16}$

Operaciones Básicas en Binario

$$\begin{array}{rcccccccc} & & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & & & \\ & 1 & 0 & 0 & 1 & . & 1 & 0 & 1_2 \\ + & 1 & 0 & 1 & 1 & . & 1 & 1 & 0_2 \\ \hline 1 & 0 & 1 & 0 & 1 & . & 0 & 1 & 1_2 \end{array}$$

$$\begin{array}{rcccccccc} & {\scriptstyle 1} & & {\scriptstyle 1} & {\scriptstyle 1} & & {\scriptstyle 1} & & \\ & 1 & 0 & 1 & 0 & 1 & . & 0 & 1 & 1_2 \\ & {\scriptstyle 1} & & {\scriptstyle 1+} & {\scriptstyle 1+} & {\scriptstyle 1+} & & & & \\ - & 1 & 0 & 1 & 1 & . & 1 & 1 & 0_2 \\ \hline 0 & 1 & 0 & 0 & 1 & . & 1 & 0 & 1_2 \end{array}$$

$$\begin{array}{rcccccccc} & & 1 & 0 & 0 & 1 & . & 1 & 0 & 1_2 \\ \times & & & & 1 & 0 & . & 1_2 & & \\ \hline & & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & & \\ & & & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & - \\ & 1 & 0 & 0 & 1 & 1 & 0 & 1 & - & \\ \hline 1 & 1 & 0 & 0 & 0. & 0 & 0 & 0 & 1_2 \end{array}$$

Operaciones Básicas en Binario (cont.)

```
               1
    1  1  0'  0'  0.' 0'  0'  0'  1_2' | 1  0.  1_2
       1+                              |------------------
                                       | 1  0  0  1.  1  0  1_2
 -  1  0  1  1   1   1                 |
 ------------                          |
    0  0  1  0   0   0                 |
          1  1+  1+                    |
       -  1  0   1   1                 |
       -------------                   |
          0  0   1   1   0             |
                     1+                |
             -   1   0   1             |
             -------------             |
                 0   0   1   0   1     |
                     -   1   0   1     |
                     -------------------
                         0   0   0     |
```

Operaciones Básicas en Octal

$$\begin{array}{r} {\scriptstyle 1\quad\quad 1} \\ 3\ 2\ 6\ .\ 0\ 5_8 \\ +\ 7\ 1\ 4\ .\ 2\ 3_8 \\ \hline 1\ 2\ 4\ 2\ .\ 3\ 0_8 \end{array}$$

$$\begin{array}{r} {\scriptstyle 1\quad\quad 1\quad\quad\ 1} \\ 1\ 2\ 4\ 2\ .\ 3\ 0_8 \\ {\scriptstyle 1\quad 1+\quad\quad 1+} \\ -\ 7\ 1\ 4\ .\ 2\ 3_8 \\ \hline 0\ 3\ 2\ 6\ .\ 0\ 5_8 \end{array}$$

$$\begin{array}{r} {\scriptstyle 1\ \ 3/2\ 1/1\quad\ 2/1} \\ 7\ 1\ 4\ .\ 2\ 3_8 \\ \times\quad\quad\quad\quad\ 5\ 6_8 \\ \hline {\scriptstyle 1\ \ 1\quad\quad\ 1} \\ 5\ 3\ 1\ 1\ 6\ 2 \\ 4\ 3\ 7\ 5\ 3\ 7\ - \\ \hline 5\ 1\ 2\ 6\ 5.\ 5\ 2_8 \end{array}$$

Operaciones Básicas en Octal (cont.)

$$
\begin{array}{rrrrrrrrr|l}
 & 5 & 1 & 2' & 6' & 5.' & 5' & 2_8' & 5\ 6_8 \\
\cline{9-9}
- & 5 & 0 & 2 & & & & & 7\ 1\ 4.\ 2\ 3_8 \\
\cline{1-4}
 & 0 & 1 & {}^{1}0 & 6 & & & & \\
 & & 1 & & & & & & \\
 & & - & 5 & 6 & & & & \\
\cline{2-5}
 & & 0 & 3 & {}^{1}0 & 5 & & & \\
 & & & 1+ & & & & & \\
 & & - & 2 & 7 & 0 & & & \\
\cline{2-6}
 & & & 0 & 1 & 5 & 5 & & \\
 & & & - & 1 & 3 & 4 & & \\
\cline{3-7}
 & & & & 0 & 2 & 1 & 2 & \\
 & & & & - & 2 & 1 & 2 & \\
\cline{5-8}
 & & & & & 0 & 0 & 0 &
\end{array}
$$

Operaciones Básicas en Hexadecimal

$$\begin{array}{rcccccc} & & {\scriptstyle 1} & & {\scriptstyle 1} & & \\ & A & 4 & 2 & . & C & E_{16} \\ + & 8 & B & 1 & . & D & 7_{16} \\ \hline 1 & 2 & F & 4 & . & A & 5_{16} \end{array}$$

$$\begin{array}{rcccccc} & {\scriptstyle 1} & & & & {\scriptstyle 1} & {\scriptstyle 1} \\ & 1 & 2 & F & 4 & . & A & 5_{16} \\ & {\scriptstyle 1} & & {\scriptstyle 1+} & & {\scriptstyle 1+} & & \\ - & 8 & B & 1 & . & D & 7_{16} \\ \hline 0 & A & 4 & 2 & . & C & E_{16} \end{array}$$

$$\begin{array}{rcccccc} & {\scriptstyle 2} & {\scriptstyle 1} & {\scriptstyle 8} & & {\scriptstyle 9} & \\ & A & 4 & 2 & . & C & E_{16} \\ \times & & & & & 1 & B_{16} \\ \hline & & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & \\ & 7 & 0 & D & E & D & A \\ & A & 4 & 2 & C & E & - \\ \hline 1 & 1 & 5 & 0 & B. & B & A_{16} \end{array}$$

Operaciones Básicas en Hexadecimal (cont.)

$$
\begin{array}{rrrrrrrrr|rrrrr}
 & & & 1 & & & & & & & & & \\
 & 1 & 1 & 5' & 0' & B.' & B & A_{16} & 1 & B_{16} & & & \\
 & & 1+ & & & & & & & A & 4 & 2. & C & E_{16} \\
- & 1 & 0 & E & 1 & & & & & & & & & \\
\hline
 & 0 & 0 & 7 & 0 & & & & & & & & & \\
 & & & 1+ & & & & & & & & & & \\
 & & - & 6 & C & & & & & & & & & \\
\hline
 & & & 0 & 4 & B & & & & & & & & \\
 & & & - & 3 & 6 & & & & & & & & \\
\hline
 & & & & 1 & 5 & B & & & & & & & \\
 & & & - & 1 & 4 & 4 & & & & & & & \\
\hline
 & & & & 0 & 1 & 7 & A & & & & & & \\
 & & & & - & 1 & 7 & A & & & & & & \\
\hline
 & & & & & 0 & 0 & 0 & & & & & &
\end{array}
$$

Acumulador

- Un acumulador es un dispositivo que posee *n* casillas para guardar en cada una un dígito en base *b*.
- Posee un mecanismo para avanzar y retroceder a la posición siguiente.
  - Ejemplos:
    - Contador de cinta de una grabadora.
    - Contador de gasolina en una estación de servicio.
    - Contador de kilómetros de un vehículo.
- Existe dos tipos de acumuladores:
  - Acumulador abierto.
  - Acumulador cerrado.

Acumulador Abierto

- Cuando se alcanza el dígito maximal en base *b* el próximo incremento en esa casilla vuelve el dígito a 0 y hace avanzar una posición la casilla a su izquierda (se produce una unidad de llevar o *carry* (acarreo)).
- Se conoce también por el nombre de **complemento a la base *b*** y **acumulador en módulo b^n**.
- Al haber *n* casillas, y cada una puede tener *b* dígitos diferentes, hay b^n posibles combinaciones de dígitos o números diferentes que se pueden representar en el acumulador.

Acumulador Abierto (cont.)

- Se llama abierto porque al ocurrir el acarreo más allá de la posición del dígito más significativo (MSB), tal unidad se pierde y el acumulador regresa a 0 en todos sus dígitos.
- De igual manera, si el acumulador está en 0, y ocurrir un retroceso, el acumulador presenta todos sus dígitos maximales.
- Se asocia a avanzar una vuelta con la operación sumar, y a retroceder una vuelta a la operación restar.

Acumulador Abierto (cont.)

- El acumulador puede representar **números sin signo** en el rango **[0,b^n – 1]**.
  - Base b = 10 y n = 3 posiciones ⇒ [0,999]
    - 000 – 999 representan los valores 0 – 999.
  - Base b = 2 y n = 3 posiciones ⇒ [0,7]
    - 000 – 111 representan los valores 0 – 7.

Acumulador Abierto (cont.)

- También, puede representar **números con signo** en el rango **[-(b^n/2),+(b^n/2 – 1)]**. Existirán b^n combinaciones de números, los cuales la mitad serán valores positivos y la otra mitad valores negativos.
  - Base b = 10 y n = 3 posiciones ⇒ [-500,499]
    - 000 – 499 representan los valores 0 – 499.
    - 500 – 999 representan los valores -500 – -1
  - Base b = 2 y n = 3 posiciones ⇒ [-4,3]
    - 000 – 011 representan los valores 0 – 3.
    - 100 – 111 representan los valores -4 – -1

Acumulador Abierto (cont.)

- Si se observa la relación entre el valor físico y el matemático, se ve que siempre hay una distancia de b^n.
  - Dos números a y b son congruentes módulo m, sii $a - b = km$, para algún entero k, y se escribe $a \equiv b \pmod{m}$.
- Ejemplo:
  - Base $b = 10$ y $n = 3$ posiciones $\Rightarrow$ [-500,499]
    - $708 \equiv -292 \bmod 1000 \Rightarrow 708 + 292 = 1000 = 1000k \Rightarrow k = 1$.
      - Entonces 708 representa en realidad a -292.

Aritmética en Acumulador Abierto

- Se debe recordar que al ocurrir un acarreo más allá de la posición del MSB, tal unidad se pierde.
- Cuando se trabaja sin signo, ocurre un desborde (*overflow*) cuando el resultado sobrepasa el límite establecido.
- Ejemplo:
  - Base $b = 10$ y $n = 3$ posiciones $\Rightarrow$ [0,999]

$$\begin{array}{cccc} & & 1 & \\ & 3 & 4 & 9 \\ + & 8 & 2 & 5 \\ \hline \cancel{1} & 1 & 7 & 4 \end{array} \Rightarrow \text{Desborde} \qquad \begin{array}{cccc} & & 1 & \\ & 1 & 5 & 4 \\ + & 8 & 2 & 6 \\ \hline 0 & 9 & 8 & 0 \end{array}$$

Aritmética en Acumulador Abierto (cont.)

- Cuando se trabaja con signo, la única manera de ocurrir un desborde (*overflow*) es que se sumen dos cantidades del mismo signo que sobrepasen el límite.
- Ejemplo:
  - Base $b = 10$ y $n = 3$ posiciones $\Rightarrow$ [-500,499]

$$\begin{array}{cccc} 1 & {}^{1}0 & {}^{1}0 & {}^{1}0 \\ {}^{1}- & {}^{1}8 & {}^{1}2 & 5 \\ \hline 0 & 1 & 7 & 5 \end{array} \qquad 825 \equiv -175$$

$$\begin{array}{cccc} & & 1 & \\ & 3 & 4 & 9 \\ + & 8 & 2 & 5 \\ \hline \cancel{1} & 1 & 7 & 4 \end{array} \qquad \begin{array}{cccc} & 3 & {}^{1}4 & 9 \\ - & {}^{1}1 & 7 & 5 \\ \hline & 1 & 7 & 4 \end{array}$$

$$\begin{array}{cccc} & & 1 & \\ & 3 & 4 & 9 \\ + & 2 & 2 & 5 \\ \hline & \textcircled{5} & 7 & 4 \end{array} \Rightarrow \text{Desborde} \qquad \begin{array}{cccc} & & 1 & \\ & 5 & 1 & 0 \\ + & 6 & 9 & 2 \\ \hline \cancel{1} & \textcircled{2} & 0 & 2 \end{array} \Rightarrow \text{Desborde}$$

Acumulador Cerrado

- Se conoce también por el nombre de **complemento a la base disminuida *b*–1** y **acumulador en módulo b^n–1**.
- Se llama cerrado porque al ocurrir el acarreo más allá de la posición del dígito más significativo (MSB), tal unidad se suma al resultado.
- Al haber *n* casillas, y cada una puede tener *b* dígitos diferentes, hay b^n posibles combinaciones de dígitos físicos o números diferentes, y b^n–1 posibles valores matemáticos diferentes, lo que implica que algún valor matemático tiene 2 representaciones.

Acumulador Cerrado (cont.)

- El acumulador puede representar **números sin signo** en el rango **[0,b^n – 1]**.
  - Base b = 10 y n = 3 posiciones ⇒ [0,999]
    - 000 – 999 representan los valores 0 – 998 con dos ceros: +0 = 000 y -0 = 999.
  - Base b = 2 y n = 3 posiciones ⇒ [0,7]
    - 000 – 111 representan los valores 0 – 6 con dos ceros: +0 = 000 (0) y -0 = 111 (7).

Acumulador Cerrado (cont.)

- También, puede representar **números con signo** en el rango **[-(b^n/2 – 1),+(b^n/2 – 1)]**. Existirán b^n combinaciones de números, los cuales la mitad serán valores positivos y la otra mitad valores negativos.
  - Base b = 10 y n = 3 posiciones ⇒ [-499,499]
    - 000 – 499 representan los valores +0 – 499.
    - 500 – 999 representan los valores -499 – -0
  - Base b = 2 y n = 3 posiciones ⇒ [-3,3]
    - 000 – 011 representan los valores +0 – 3.
    - 100 – 111 representan los valores -3 – -0

Acumulador Cerrado (cont.)

- Se observa que el valor matemático hay siempre una distancia de b^n–1.
  - Dos números a y b son congruentes módulo m, sii $a - b = km$, para algún entero k, y se escribe $a \equiv b \pmod{m}$.
- Ejemplo:
  - Base $b = 10$ y $n = 3$ posiciones $\Rightarrow$ [-499,499]
    - $708 \equiv -291 \bmod 999 \Rightarrow 708 + 291 = 999 = 999k \Rightarrow k = 1$.
      - Entonces 708 representa en realidad a -291.

Aritmética en Acumulador Cerrado

- Se debe recordar que al ocurrir un acarreo más allá de la posición del MSB, tal unidad se suma al resultado.
- Cuando se trabaja sin signo, ocurre un desborde (*overflow*) cuando el resultado sobrepasa el límite establecido.
- Ejemplo:
  - Base $b = 10$ y $n = 3$ posiciones $\Rightarrow$ [0,999]

$$\begin{array}{r} {\scriptstyle 1} \\ 3\ 7\ 1 \\ +\ 7\ 4\ 1 \\ \hline 1\ 1\ 1\ 2 \\ \llcorner\!\!+\ \ \ 1 \\ \hline 1\ 1\ 3 \end{array} \Rightarrow \text{Desborde} \qquad \begin{array}{r} 2\ 5\ 4 \\ +\ 7\ 4\ 2 \\ \hline 9\ 9\ 6 \end{array}$$

Aritmética en Acumulador Cerrado (cont.)

- Cuando se trabaja con signo, ocurre un desborde (*overflow*) al igual que en el acumulador abierto.
- Ejemplo:
  - Base $b = 10$ y $n = 3$ posiciones $\Rightarrow$ [-499,499]

$$\begin{array}{rrrr} & 9 & 9 & 9 \\ - & 8 & 2 & 5 \\ \hline & 1 & 7 & 4 \end{array} \qquad 825 \equiv -174$$

$$\begin{array}{rrrr} & & 1 & \\ & 3 & 4 & 9 \\ + & 2 & 2 & 5 \\ \hline & (5) & 7 & 4 \end{array} \Rightarrow \text{Desborde}$$

$$\begin{array}{rrrr} & 3 & 4 & 9 \\ + & 8 & 2 & 5 \\ \hline 1 & 1 & 7 & 4 \\ \hookrightarrow & + & & 1 \\ \hline & 1 & 7 & 5 \end{array}$$

$$\begin{array}{rrrr} & & 1 & \\ & 5 & 1 & 0 \\ + & 6 & 9 & 2 \\ \hline 1 & 2 & 0 & 2 \\ \hookrightarrow & + & & 1 \\ \hline & (2) & 0 & 3 \end{array} \Rightarrow \text{Desborde}$$

$$\begin{array}{rrrr} & 3 & {}^{1}4 & 9 \\ - & {}^{1}1 & 7 & 4 \\ \hline & 1 & 7 & 5 \end{array}$$

Representación de Enteros – Sin Signo

- Para representar un número entero sin signo primero se pasa a binario y luego se puede almacenar en:
  - Un byte = 8 bits.
  - Una palabra = 2 bytes.
  - Una doble palabra = 4 bytes.
  - Una palabra cuádruple = 8 bytes.
- Ejemplo:
  - $200_{10} = 11001000_2 \Rightarrow$ Se almacena en un byte
  - $345_{10} = 101011001_2 \Rightarrow$ Se almacena en una palabra
  - $65712_{10} = 1000000010110000_2 \Rightarrow$ Se almacena en una doble palabra

Representación de Enteros – Con Signo

- Para representar un número entero con signo, se debe indicar el signo ya sea por 0(+) ó 1(-).
- Se han utilizado diferentes formas para distinguir entre números positivos y negativos, tres de estos métodos son:
  - Signo y magnitud.
  - Complemento a 1 = Complemento a la base disminuida.
  - Complemento a 2 = Complemento a la base.

Representación de Enteros – Con Signo
Signo y Magnitud

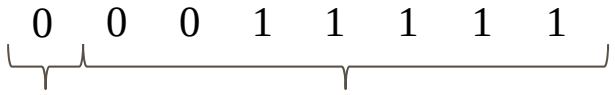
- Este es el método más simple.
  - El bit más significativo (MSB), el de más a la izquierda, representa el signo: 0 para positivo y 1 para negativo.
  - El resto de los bits representan la magnitud.
- Este método contiene un número igual de enteros positivos y negativos.
  - Un entero en signo y magnitud de n bits está en el rango $[-(2^{n-1}-1),+(2^{n-1}-1)]$, con dos posibles representaciones del cero.

Representación de Enteros – Con Signo
Signo y Magnitud (cont.)

- Ejemplo:
  - En un byte se puede representar un rango: [-127,+127].
  - 31_{10} puede ser almacenado en un byte, donde 1 bit es de signo y 7 bits son la magnitud:

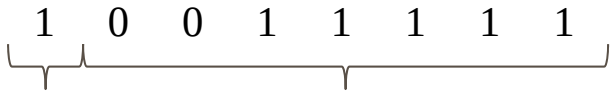
$31_{10} = 11111_2$

+31



Signo Magnitud

-31



Signo Magnitud

Representación de Enteros – Con Signo
Signo y Magnitud (cont.)

- Para sumar se hace lo siguiente:
  - Si los números tiene igual signo, se suman las magnitudes y se pone el mismo bit de signo al resultado.
  - Si los números tienen diferente signo, se comparan las magnitudes, restar la más pequeña a la más grande y dar el resultado el bit de signo de la más grande.
- Todos estos “si”, “sumas”, “restas” y “comparaciones” se traducen en circuitos bastantes complejos.
- Puede ocurrir desborde al sumar números con el mismo signo cuyo resultado sobrepase el rango establecido.

Representación de Enteros – Con Signo
Signo y Magnitud (cont.)

- Ejemplo:

$$\begin{array}{rr} & \\ +3 & 1 \\ +\ +8 & 5 \\ \hline +1 & 1\ \ 6 \end{array} \Rightarrow \begin{array}{rr} & \\ & 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1 \\ + & 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\ \hline & 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0 \end{array}$$

$$\begin{array}{rr} & \\ +8 & 5 \\ +\ -3 & 1 \\ \hline +\ 5 & 4 \end{array} \Rightarrow \begin{array}{rr} & \\ & 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\ - & 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1 \\ \hline & 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0 \end{array}$$

$$\begin{array}{rr} & \\ -3 & 1 \\ +\ -8 & 5 \\ \hline -1 & 1\ \ 6 \end{array} \Rightarrow \begin{array}{rr} & \\ & 1\ 0\ 0\ 1\ 1\ 1\ 1\ 1 \\ + & 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\ \hline & 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0 \end{array}$$

$$\begin{array}{rr} & \\ -8 & 5 \\ +\ +3 & 1 \\ \hline -\ 5 & 4 \end{array} \Rightarrow \begin{array}{rr} & \\ & 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\ + & 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1 \\ \hline & 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0 \end{array}$$

Representación de Enteros – Con Signo
Signo y Magnitud (cont.)

- Ejemplo:

						1	1	1	1	1	1								
+1	2	7			0	1	1	1	1	1	1	1							
+		+1	⇒	+	0	0	0	0	0	0	0	1							
+1	2	8			0	0	0	0	0	0	0	0	=	+0	⇒	Desborde			

						1	1	1	1	1	1								
-1	2	7			1	1	1	1	1	1	1	1							
+		-1	⇒	+	1	0	0	0	0	0	0	1							
-1	2	8			1	0	0	0	0	0	0	0	=	-0	⇒	Desborde			

Representación de Enteros – Con Signo Complemento

- Mientras que los sistemas de signo y magnitud niegan un número al cambiar su signo, el método del complemento niega un número al tomar su complemento definido por el sistema.
- Obtener el complemento es un poco más difícil que cambiar el signo, pero los dos números complementados pueden sumarse o restarse directamente sin verificar el signo y la magnitud.
  - Es mucho más simple y fácil.
- Al igual que el de signo y magnitud, se trabaja con un número fijo de dígitos.
- Si un número *D* se complementa dos veces el resultado es *D*.

	Complemento			
Dígito	**Binario**	**Octal**	**Decimal**	**Hexadecimal**
0	1	7	9	F
1	0	6	8	E
2	-	5	7	D
3	-	4	6	C
4	-	3	5	B
5	-	2	4	A
6	-	1	3	9
7	-	0	2	8
8	-	-	1	7
9	-	-	0	6
A	-	-	-	5
B	-	-	-	4
C	-	-	-	3
D	-	-	-	2
E	-	-	-	1
F	-	-	-	0

Representación de Enteros – Con Signo Complemento a la Base Disminuida

- Esto se explicó anteriormente, ya que es el acumulador cerrado.
- Rango: $[-(b^n/2 - 1),+(b^n/2 - 1)]$.
- El MSB es el bit que indica el signo: 0 para positivo y 1 para negativo.
- Hay dos representaciones del cero: +0 y -0.

Representación de Enteros – Con Signo Complemento a la Base Disminuida (cont.)

- Para representar un número en forma de complemento a la base disminuida se hace lo siguiente:
  - Verificar cuál será su tamaño para representarlo y conocer la cantidad mínima de dígitos necesarios.
  - Complementar el número, si es positivo sólo se hace la conversión a la base con la que se está trabajando, y si es negativo se hace mediante dos formas:
    - Restando el número a la base disminuida (b^n–1) y realizando la conversión del resultado a la base b.
    - Poniendo para cada dígito su dígito complemento (ver tabla de complementos de dígitos).

Representación de Enteros – Con Signo Complemento a la Base Disminuida (cont.)

- Ejemplo:
  - Base 10 $\Rightarrow$ Complemento a 9.
  - Rango: [-49,+49].
  - 31_{10} puede ser representado en 2 dígitos:

31_{10} $\quad b = 100 \quad \Rightarrow b-1 = 99$

$+31 = 3\ 1$ $\qquad -31 = 6\ 8$

$$\begin{array}{rcc} (1) & 9 & 9 \\ - & 3 & 1 \\ \hline & 6 & 8 \end{array}$$

(2) $3\ 1 \Rightarrow 6\ 8$

Representación de Enteros – Con Signo Complemento a la Base Disminuida (cont.)

- Ejemplo:
  - Base 2 $\Rightarrow$ Complemento a 1.
  - Rango: [-127,+127].
  - 31_{10} puede ser representado en 8 dígitos:

31_{10} $\quad b = 100000000 \Rightarrow b\text{-}1 = 11111111$

+31	=	0	0	0	1	1	1	1	1

-31	=	1	1	1	0	0	0	0	0
(1)		1	1	1	1	1	1	1	1
	-	0	0	0	1	1	1	1	1
		1	1	1	0	0	0	0	0
(2)		0	0	0	1	1	1	1	1
	$\Rightarrow$	1	1	1	0	0	0	0	0

Representación de Enteros – Con Signo Complemento a la Base

- Esto se explicó anteriormente, ya que es el acumulador abierto.
- Rango: $[-(b^n/2),+(b^n/2 - 1)]$.
- El MSB es el bit que indica el signo: 0 para positivo y 1 para negativo.
- Hay una representación del cero: +0.

Representación de Enteros – Con Signo Complemento a la Base (cont.)

- Para representar un número en forma de complemento a la base disminuida se hace lo siguiente:
  - Verificar cuál será su tamaño para representarlo y conocer la cantidad mínima de dígitos necesarios.
  - Complementar el número, si es positivo sólo se hace la conversión a la base con la que se está trabajando, y si es negativo se hace mediante dos formas:
    - Restando el número a la base (b^n) y realizando la conversión del resultado a la base b.
    - Poniendo para cada dígito su dígito complemento (ver tabla de complementos de dígitos) y sumar 1 al resultado.

Representación de Enteros – Con Signo Complemento a la Base (cont.)

- Ejemplo:
  - Base 10 $\Rightarrow$ Complemento a 10.
  - Rango: [-50,+49].
  - 31_{10} puede ser representado en 2 dígitos:

31_{10} $\quad b = 100$

$+31 = 3\ 1$ $\qquad -31 = 6\ 9$

(1)	1	$^{1}0$	$^{1}0$
	$^{1}-$	$^{1+}3$	1
	0	6	9

(2) $3\ 1 \Rightarrow$

	6	8
+		1
6		9

Representación de Enteros – Con Signo Complemento a la Base (cont.)

- Ejemplo:
  - Base 2 $\Rightarrow$ Complemento a 2.
  - Rango: [-128,+127].
  - 31_{10} puede ser representado en 8 dígitos:

31_{10}		b	=	100000000					
+31	=	0	0	0	1	1	1	1	1

-31	=	1	1	1	0	0	0	0	0
(1)		1	$^{1}0$	$^{1}0$	$^{1}0$	$^{1}0$	$^{1}0$	$^{1}0$	$^{1}0$	$^{1}0$
		$^{1}-$	$^{1+}0$	$^{1+}0$	$^{1+}0$	$^{1+}1$	$^{1+}1$	$^{1+}1$	$^{1+}1$	1
		0	1	1	1	0	0	0	0	1
(2)			0	0	0	1	1	1	1	1
	$\Rightarrow$		1	1	1	0	0	0	0	0
			+							1
			1	1	1	0	0	0	0	1

Tabla Resumen de las Reglas de Suma y Resta para Números Enteros Binarios

Sistema de Números	Reglas de Suma	Reglas de Negación	Reglas de Resta
Sin signo	Se suman los números. El resultado está fuera de rango (desborde), si ocurre un acarreo fuera del MSB.	N/A	Se resta el sustraendo del minuendo. El resultado está fuera del rango (desborde), si ocurre un préstamo fuera del MSB.

Tabla Resumen de las Reglas de Suma y Resta para Números Enteros Binarios

Sistema de Números	Reglas de Suma	Reglas de Negación	Reglas de Resta
Signo y magnitud	Igual signo. Se suman las magnitudes; se da un desborde si ocurre un acarreo fuera del MSB. Diferente signo. Se restan la magnitud más pequeña a la mayor; un desborde es imposible; el resultado tiene el signo de la magnitud mayor.	Cambie el signo de bit.	Cambie el bit de signo del sustraendo y proceda como una suma.

Tabla Resumen de las Reglas de Suma y Resta para Números Enteros Binarios

Sistema de Números	Reglas de Suma	Reglas de Negación	Reglas de Resta
Complemento a 2	Se suma e ignora cualquier acarreo fuera del MSB. El desborde ocurre si los acarreos entrante y saliente al MSB son diferentes.	Se complementan todos los bits del sustraendo; se suma 1 al resultado.	Se complementan todos los bits del sustraendo, se suma 1 al resultado; y se procede como en la suma.
Complementos a 1	Se suma y si hay un acarreo fuera del MSB, se suma 1 al resultado. El desborde ocurre si los acarreos entrante y saliente al MSB son diferentes.	Se complementan todos los bits del sustraendo.	Se complementan todos los bits del sustraendo y se procede como en la suma.

Códigos Binarios de Números Decimales

- Los números binarios son los más apropiados para los cálculos internos en un sistema digital, pero la mayoría de la gente todavía prefiere trabajar con los números decimales.
- Como resultado, las interfaces externas de un sistema digital pueden leer o exhibir números decimales.
- Un conjunto de cadenas de *n* bits en que las diferentes cadenas de bits representan diferentes números u otras cosas se llama **código**.
- Una combinación particular de valores de *n* bits se llama **palabra del código**.

Códigos Binarios de Números Decimales (cont.)

- Puede que en un código haya o no una relación aritmética entre los valores de los bits en una palabra de código y lo que representan.
- Además, un código que usa cadenas de *n* bits no necesita contener 2^n palabras de código.
- Al menos se necesitan 4 bits para representar los diez dígitos decimales.
  - Hay muchas maneras diferentes para elegir las 10 palabras código de 4 bits, los más comunes se muestran en la siguiente tabla.

Códigos Binarios de Números Decimales (cont.)

Dígito Decimal	BCD (8421)	Aiken (2421)	Exceso 3	Biquinario	1 de 10
0	0000	0000	0011	0100001	1000000000
1	0001	0001	0100	0100010	0100000000
2	0010	0010	0101	0100100	0010000000
3	0011	0011	0110	0101000	0001000000
4	0100	0100	0111	0110000	0000100000
5	0101	1011	1000	1000001	0000010000
6	0110	1100	1001	1000010	0000001000
7	0111	1101	1010	1000100	0000000100
8	1000	1110	1011	1001000	0000000010
9	1001	1111	1100	1010000	0000000001

Códigos Binarios de Números Decimales (cont.)

Palabras de código no usadas				
BCD (8421)	**Aiken (2421)**	**Exceso 3**	**Biquinario**	**1 de 10**
1010	0101	0000	0000000	0000000000
1011	0110	0001	0000001	0000000011
1100	0111	0010	0000010	0000000101
1101	1000	1101	0000011	0000000110
1110	1001	1110	0000101	0000000111
1111	1010	1111	…	…

Códigos Binarios de Números Decimales – Código BCD

- El código decimal más natural es el BCD (*binary-coded decimal*), que codifica los dígitos 0 a 9 por sus representaciones binarias sin signo en 4 bits, del 0000 al 1001.
- En un byte caben dos dígitos en BDC.
- Los códigos BCD más usados son:
  - Natural (8421).
  - Aiken (2421).
  - 5421.
  - Exceso 3.

Códigos Binarios de Números Decimales – Código BCD (cont.)

- En el BCD sólo se utilizan 10 de las 16 posibles combinaciones que se pueden formar con números de 4 bits, por lo que el sistema pierde capacidad de representación, aunque se facilita la compresión de los números.
  - El BCD solo se usa para representar **cifras** no números en su totalidad.
  - Esto quiere decir que para números de más de una cifra hacen falta dos números BCD para componerlo.
- Los pesos para los bits BDC son 8, 4, 2 y 1, por esta razón se le llama código 8421.

Códigos Binarios de Números Decimales – Código BCD (cont.)

- Desde que los sistemas informáticos empezaron a almacenar los datos en conjuntos de ocho bits, hay dos maneras comunes de almacenar los datos BCD:
  - Omisión de los cuatro bits más significativos(como sucede en el EBCDIC).
  - Almacenamiento de dos datos BCD, es el denominado BCD "empaquetado", en el que también se incluye en primer lugar el signo, por lo general con 1100 para el + y 1101 para el -.
- De este modo, el número 127 sería representado como (11110001, 11110010, 11110111) en el EBCDIC o (00010010, 01111100) en el BCD empaquetado

Códigos Binarios de Números Decimales – Código BCD (cont.)

- El BCD sigue siendo ampliamente utilizado para almacenar datos, en aritmética binaria o en electrónica. Los números se pueden mostrar fácilmente en visualizadores de siete segmentos enviando cada cuarteto BCD a un visualizador.
- La BIOS de un ordenador personal almacena generalmente la fecha y la hora en formato del BCD, probablemente por razones históricas se evitó la necesidad de su conversión en ASCII.

Códigos Binarios de Números Decimales – Código BCD (cont.)

- La ventaja del código BCD frente a la representación binaria clásica es que no hay límite para el tamaño de un número.
- Los números que se representan en formato binario están generalmente limitados por el número mayor que se pueda representar con 8, 16, 32 o 64 bits.
- Por el contrario utilizando BCD añadir un nuevo dígito sólo implica añadir una nueva secuencia de 4 bits.

Códigos Binarios de Números Decimales – Código BCD (cont.)

- La suma de dígitos BCD es similar a la suma de números binarios sin signo, excepto que se debe hacerse una corrección si el resultado excede 1001.
  - El resultado se corrige sumándole 6.

```
   5          0101          5          0101
 + 4        + 0100        + 9        + 1001
 ---        ------        ---        ------
   9          1001         14          1110
                                    + 0110  – Corrección
                                    ------
   5        0000 0101    10+4       1 0100
 + 11     + 0001 0001
 ---      -----------
  16        0001 0110
```

Códigos Binarios de Números Decimales – Código BCD (cont.)

- Otro conjunto de pesos resulta el **código 2421**, que es un *código autocomplementario*; o sea, la palabra código para el complemento a 9 de cualquier dígito puede obtenerse al complementar los bits individuales de la palabra código del dígito.
- El **código de exceso 3** es otro código autocomplementario, tiene una relación aritmética con el BDC; ya que la palabra código para cada dígito decimal es la correspondiente a la de BCD más 0011 (+3).

Códigos Binarios de Números Decimales – Código BCD (cont.)

- Los códigos decimales pueden tener más de 4 bits, como el **código biquinario** que usa 7 bits.
  - Los primeros dos bits en una palabra código indican si el número está en el rango 0-4 o 5-9 y los últimos 5 bits indican cuál de los cinco números del rango seleccionado está representado.
- El **código 1 de 10** es la codificación menos densa para los dígitos decimales, usando sólo 10 palabras de código de las 1024 posibles.

Códigos Binarios de Números Decimales – Código BCD (cont.)

- El término ***biquinario*** se refiere a que el código tiene una parte de dos estados (*bi*) y otra de cinco estados (*quin*).
- Existen varias representaciones de un decimal codificado en biquinario, ya que:
  - El componente de dos estados se puede representar tanto con uno como con dos bits.
  - El componente de cinco estados, tanto con tres como con cinco bits.

Código Gray

- Este es un código binario no ponderado y tiene la propiedad de que los códigos para dígitos decimales sucesivos difiere en un sólo bit, al código Gray también se le llama autorreflejado o cíclico.
  - Es un caso particular de sistema binario.
- Consiste en una ordenación de 2^n números binarios de tal forma que cada número sólo tenga un dígito binario distinto a su predecesor.
- Este código puede representar números o cosas.

Código Gray (cont.)

- Historia:
  - Esta técnica de codificación se originó cuando los circuitos lógicos digitales se realizaban con válvulas de vacío y dispositivos electromecánicos.
  - Los contadores necesitaban potencias muy elevadas a la entrada y generaban picos de ruido cuando varios bits cambiaban simultáneamente.
  - El uso de código Gray garantizó que en cualquier transición variaría tan sólo un bit.

Código Gray (cont.)

- El primer uso documentado de un código de estas características fue en una demostración del telégrafo del ingeniero francés Émile Baudot, en 1878.
- Pero no fueron patentados hasta 1953 por Frank Gray (que dio nombre al sistema de codificación), un investigador de los laboratorios Bell.
- Hay varios algoritmos para generar una secuencia de código Gray (y varios códigos posibles resultantes, en función del orden que se desee seguir), pero el más usado consiste en cambiar el bit menos significativo que genera un nuevo código.

Código Gray (cont.)

Número Decimal	Código Binario	Código Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

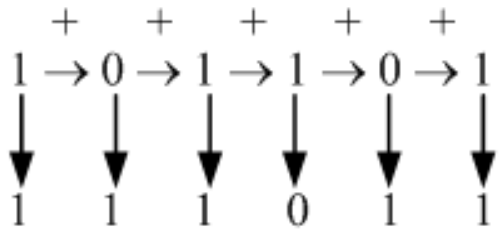
Código Gray (cont.)

Número Decimal	Código Binario	Código Gray
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Código Gray (cont.)

- Para convertir de Binario a Gray puede seguirse el siguiente procedimiento:
  - El MSB se deja igual.
  - Avanzando de MSB a LSB se suma cada bit con el siguiente despreciando el acarreo para obtener el siguiente bit del código Gray.
- Ejemplo: Pasar el número decimal 45 a código Gray.

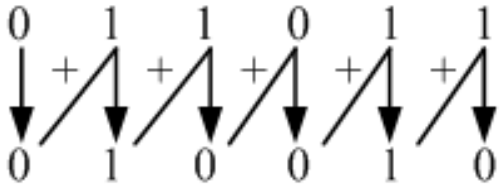
$$45_{10} = 101101_2$$



$$45_{10} = 111011_{gray}$$

Código Gray (cont.)

- Para convertir de Gray a Binario puede seguirse el siguiente procedimiento:
  - El MSB se deja igual.
  - Avanzando de MSB a LSB a cada bit obtenido en binario se le suma sin acarreo el siguiente bit de código Gray.
- Ejemplo: Obtener el equivalente decimal del siguiente código gray 011011_{gray}.



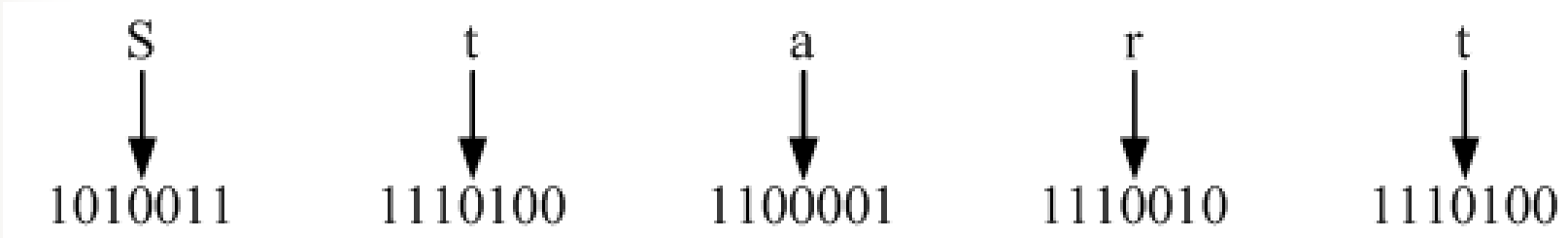
$$011011_{gray} = 010010_2$$
$$010010_2 = 18_{10}$$

Códigos de Caracteres o Alfanuméricos

- Muchas aplicaciones de sistemas digitales (especialmente las computadoras o la transmisión de textos) requieren del procesamiento de datos los como números, letras y símbolos especiales.
- Para manejar estos datos usando dispositivos digitales, cada símbolo debe estar representado por un código binario.
- El código alfanumérico más generalizado en la actualidad es el denominado ASCII (*American Standard Code for Information Interchange*). Este es un código de 7 bits.

Códigos de Caracteres o Alfanuméricos (cont.)

- Ver http://www.isa.cie.uva.es/proyectos/codec/teoria2.html.
- Ejemplo: La palabra "Start" se representa en código ASCII como sigue:



Códigos de Detección y Corrección de Errores

- Los sistemas digitales pueden cometer errores de vez en cuando.
- Aunque los dispositivos en circuito integrado que carecen de partes móviles tienen alta confiabilidad; pero los dispositivos que tienen interacción con partes móviles son menos confiables.
- Cuando se leen, escriben o transmiten caracteres de un sitio a otro, pueden producirse errores

Códigos de Detección y Corrección de Errores (cont.)

- Ejemplos:
  - Se pueden producir errores por polvo en las cabezas lectoras de una unidad de disco.
  - La ocurrencia de errores en la *transmisión de datos a distancia*.
  - Los datos que se transmiten por modem pueden recibirse incorrectamente si la línea tiene ruidos.
  - Las perturbaciones en el suministro de energía eléctrica pueden producir errores.
- En esta sección se ilustran dos códigos que permiten detectar errores y, en algunos casos, incluso corregirlos.

Códigos de Detección y Corrección de Errores – Código de Paridad

- Un método muy simple, pero ampliamente utilizado por su sencillez para detectar errores en transmisión de datos consiste en añadir un **bit de paridad a cada carácter**, normalmente en la posición más significativa.
  - En el código de **paridad par**, el bit de paridad se elige de manera que el número de bits 1 del dato sea un número par incluyendo el bit de paridad.
  - En el código de **paridad impar**, el bit de paridad se elige de modo que el número de bits 1 (incluyendo el de paridad) del dato sea impar.

Códigos de Detección y Corrección de Errores – Código de Paridad (cont.)

Bit de Paridad	Código ASCII							Carácter
0	1	0	1	0	0	1	1	S
0	1	1	1	0	1	0	0	t
1	1	1	0	0	0	0	1	a
0	1	1	1	0	0	1	0	r
0	1	1	1	0	1	0	0	t

Códigos de Detección y Corrección de Errores – Código de Paridad (cont.)

Bit de Paridad	Código ASCII							Carácter
1	1	0	1	0	0	1	1	S
1	1	1	1	0	1	0	0	t
0	1	1	0	0	0	0	1	a
1	1	1	1	0	0	1	0	r
1	1	1	1	0	1	0	0	t

Códigos de Detección y Corrección de Errores – Código de Paridad (cont.)

- De esta manera, cuando cambia un bit durante la transmisión, el número de unos en el carácter recibido tendrá la paridad equivocada y el receptor sabrá que se ha producido un error.
- Ejemplo:
  - Si un transmisor envía “Start” y hay errores en la transmisión, suponiendo que el receptor recibe la siguiente información, en la siguiente tabla se anota los datos que llegaron erróneos y si se detectó o no el error, agrega en la columna vacía cuantos bits cambiaron en la transmisión.

Códigos de Detección y Corrección de Errores – Código de Paridad (cont.)

Dato Enviado	Dato Recibido (supuesto)	Error	Paridad	Bits Erróneos
01010011 (S)	01010010 (R)	Sí	Mal	1
01110100 (t)	01100010 (>)	Sí	Mal	3
11100001 (a)	11100001 (a)	No	Bien	0
01110010 (r)	01110001 (G)	Sí	Bien	2
01110100 (t)	01110100 (t)	No	Bien	0

Códigos de Detección y Corrección de Errores – Código de Paridad (cont.)

Dato Enviado	Dato Recibido (supuesto)	Error	Paridad	Bits Erróneos
11010011 (S)	11010010 (R)	Sí	Mal	1
11110100 (t)	11100010 (>)	Sí	Mal	3
01100001 (a)	01100001 (a)	No	Bien	0
11110010 (r)	11110001 (G)	Sí	Bien	2
11110100 (t)	11110100 (t)	No	Bien	0

Códigos de Detección y Corrección de Errores – Código de Paridad (cont.)

- Como puede verse, el código de paridad No siempre resulta efectivo para detectar errores.
- ¿Qué tipo de errores si detecta y cuales no?
  - Detecta cuando hay una cantidad impar de errores.
  - Cuando hay una cantidad par de errores no detecta nada.
- Este código sólo detecta errores en una cantidad impar de bits, no corrige.

Códigos de Detección y Corrección de Errores – Código de Hamming

- Richard Hamming (1950) ideó un método no sólo para detectar errores sino también para corregirlos, y se conoce como **código Hamming.**
- Se añaden *k* bits de paridad a un carácter de *n* bits, formando un nuevo carácter de $n + k$ bits. Los bits se numeran empezando por 1, no por 0, siendo el bit 1 el MSB.
- Todo bit cuyo número sea potencia de 2 es un bit de paridad y todos los demás se utilizan para datos.
- Para un carácter ASCII de 7 bits, se añaden 4 bits de paridad.
  - Los bits 1, 2, 4 y 8 son bits de paridad; 3, 5, 6, 7, 9, 10 y 11 son los 7 bits de datos.

Códigos de Detección y Corrección de Errores – Código de Hamming (cont.)

- Cada bit de paridad comprueba determinadas posiciones de bit y se ajusta de modo que el número total de unos en las posiciones comprobadas sea par, si se trata de paridad par; o sea impar, si se trata de paridad impar.
- En este código se pueden detectar errores en 1 o 2 bits, y también corregir errores en un solo bit.
  - Esto representa una mejora respecto a los códigos con bit de paridad, que pueden detectar errores en sólo un bit, pero no pueden corregirlo.

Códigos de Detección y Corrección de Errores – Código de Hamming (cont.)

- Las posiciones de los bits comprobados por los de paridad son:
  - El bit 1 comprueba los bits 1, 3, 5, 7, 9 y 11.
  - El bit 2 comprueba los bits 2, 3, 6, 7, 10 y 11.
  - El bit 4 comprueba los bits 4, 5, 6 y 7.
  - El bit 8 comprueba los bits 8, 9, 10 y 11.
- En general, el bit n es comprobado por los bits b_1, b_2,....,b_j, tales que $b_1 + b_2 + ... + b_j = n$.
  - Por ejemplo:
    - El bit 5 es comprobado por los bits 1 y 4 porque 1 + 4 = 5.
    - El bit 6 es comprobado por los bits 2 y 4 porque 2 + 4 = 6.

Códigos de Detección y Corrección de Errores – Código de Hamming (cont.)

- Ejemplo: Usando paridad par, construir el código de Hamming para el carácter "b“.

Código ASCII para “b”:

1	1	0	0	0	1	0

Código de Hamming para “b”:

1	2	3	4	5	6	7	8	9	10	11
		1		1	0	0		0	1	0

0	0	1	1	1	0	0	1	0	1	0

Códigos de Detección y Corrección de Errores – Código de Hamming (cont.)

- Considérese que pasaría si el bit 1 se modificara durante la transmisión.
  - El carácter recibido sería 10111001010 en lugar de 00111001010.
  - El receptor comprobaría los 4 bits de paridad con los resultados siguientes:
    - Bit de paridad 1 incorrecto: bits 1, 3, 5, 7, 9 y 11 contienen tres unos.
    - Bit de paridad 2 correcto: los bits 2, 3, 6, 7, 10 y 11 contienen dos unos.
    - Bit de paridad 4 correcto: los bits 4, 5, 6 y 7 contienen dos unos.
    - Bit de paridad 8 correcto: los bits 8, 9, 10 y 11 contienen dos unos.

Códigos de Detección y Corrección de Errores – Código de Hamming (cont.)

1	2	3	4	5	6	7	8	9	10	11
1	0	1	1	1	0	0	1	0	1	0
1	**0**	1	1	1	0	0	1	0	1	0
1	0	1	**1**	1	0	0	1	0	1	0
1	0	1	1	1	0	0	**1**	0	1	0

Códigos de Detección y Corrección de Errores – Código de Hamming (cont.)

- El número total de unos en los bits 1, 3, 5, 7, 9 y 11 debería de ser par, ya que se está usando paridad par.
- El bit incorrecto debe ser uno de los bits comprobados por el bit de paridad 1, es decir, uno de los bits 1, 3, 5, 7, 9 u 11.
  - El bit de paridad 2 es correcto, sabemos que los bits 2, 3, 6, 7, 10 y 11 son correctos, de forma que el error no estaba en los bits 3, 7 u 11. Esto deja los bits 1, 5 y 9.
  - El bit de paridad 4 es correcto, lo cual significa que los bits 4, 5, 6 y 7 no contienen errores. Esto reduce la elección al 1 ó 9.
  - El bit de paridad 8 también es correcto y, por lo tanto, el bit 9 es correcto. Por consiguiente, el bit incorrecto debe ser el 1.
  - Dado que se recibió como un 1, debería haberse transmitido como un 0. En esta forma se pueden corregir los errores

Código de Huffman

- Código óptimo dentro de los códigos de codificación estadística, es el código de menor longitud media.
- A los símbolos con mayor frecuencia de aparición se les asignarán las palabras de código binario de menor longitud.
  - Se ordena el conjunto de símbolos del alfabeto fuente en orden creciente de probabilidades de aparición.
  - Se juntan los dos símbolos con menor probabilidad de aparición en un único símbolo, cuya probabilidad será la suma de las probabilidades de los símbolos que lo originaron.
  - Se repite este proceso hasta que sólo tengamos dos símbolos.

Código de Huffman (cont.)

- Se asigna un 1 a uno de los dos símbolos que tenemos y un 0 al otro.
- Recorreremos la estructura que hemos construido hacia atrás, cuando dos símbolos hayan dado origen a un nuevo símbolo, estos "heredarán" la codificación asignada a este nuevo símbolo.
- Se le añadirá un 1 a la codificación de uno de los símbolos y un 0 a la del otro símbolo.
- Sustituimos cada palabra del texto por el código respectivo y, una vez hecho esto, agrupamos los bits en grupos de ocho, es decir en bytes.

Ejemplo del Código de Huffman

- Se obtienen las frecuencias de cada palabra dentro del documento:

casa	29
nuevo	7
pesa	12
plato	5
sucio	4
tarde	8

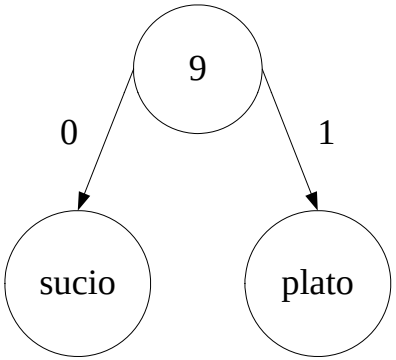
- Se ordenan las frecuencias en orden ascendente:

(sucio, plato, nuevo, tarde, pesa, casa)

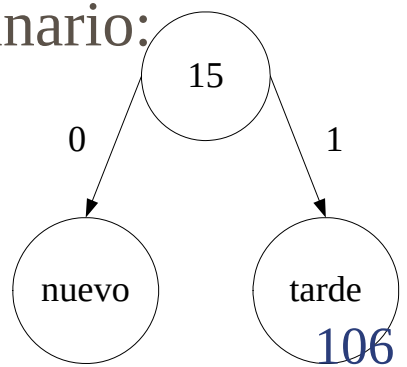
(4, 5, 7, 8, 12, 29)

Ejemplo del Código de Huffman (cont.)

- Luego se eligen los dos valores más pequeños y se construye un árbol binario con hojas etiquetadas:

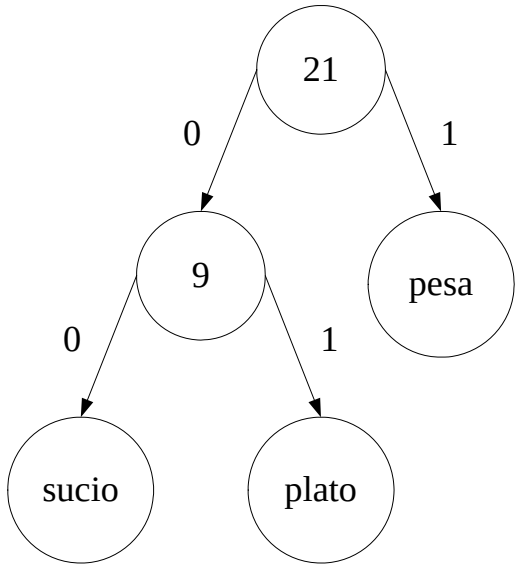


- Se reemplazan los dos valores por su suma, obteniéndose una nueva secuencia (7, 8, 9, 12, 29). De nuevo, se toman los dos valores más pequeños y se construye el árbol binario:



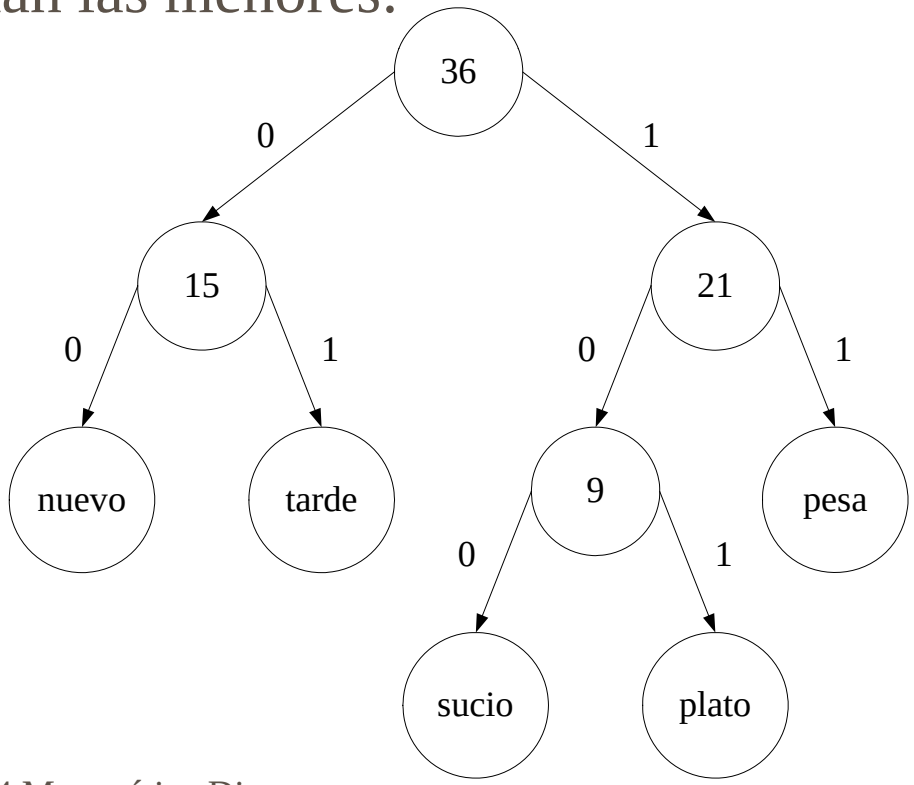
Ejemplo del Código de Huffman (cont.)

- Ahora se tienen las frecuencias (9, 12, 15, 29) y una vez más se seleccionan las menores:



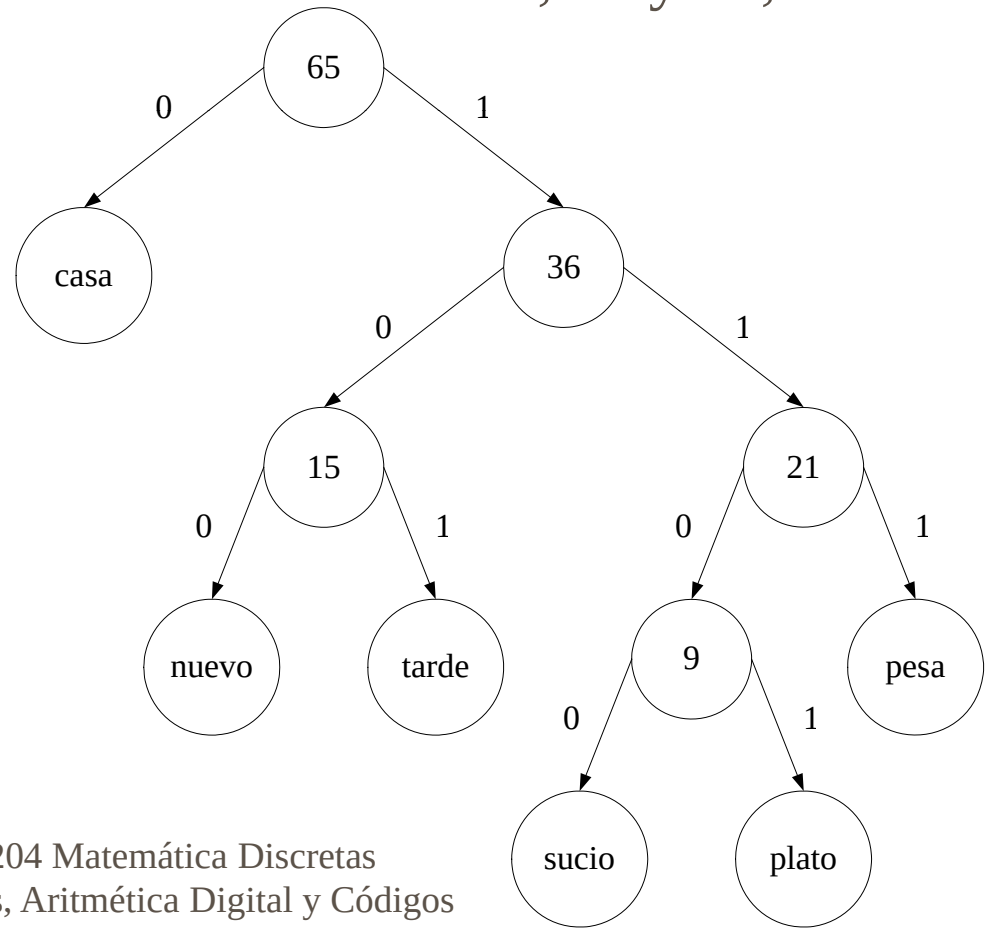
Ejemplo del Código de Huffman (cont.)

- Ahora se tienen las frecuencias (15, 21, 29) y una vez más se seleccionan las menores:



Ejemplo del Código de Huffman (cont.)

- Las dos frecuencias restantes, 29 y 36, se combinan en el árbol final:



Ejemplo del Código de Huffman (cont.)

- Del árbol anterior obtenemos el código para este alfabeto:

casa	0
nuevo	100
pesa	111
plato	1101
sucio	1100
tarde	101

- Sustituimos cada palabra del texto por el código respectivo y, una vez hecho esto, agrupamos los bits en grupos de ocho, es decir en bytes.

Referencias Bibliográficas

- Jonnsonbaugh, Richard. “Matemáticas Discretas”. Prentice Hall, México. Sexta Edición, 2005.
- Elizande, María Guadalupe. “Introducción a los Sistemas Computacionales”. URL: http://www.fismat.umich.mx/~elizalde/curso/curso.html.
- Código Binario Decimal. URL: http://es.wikipedia.org/wiki/C%C3%B3digo_binario_decimal.
- Tablas de Códigos. URL: http://www.isa.cie.uva.es/proyectos/codec/teoria2.html.