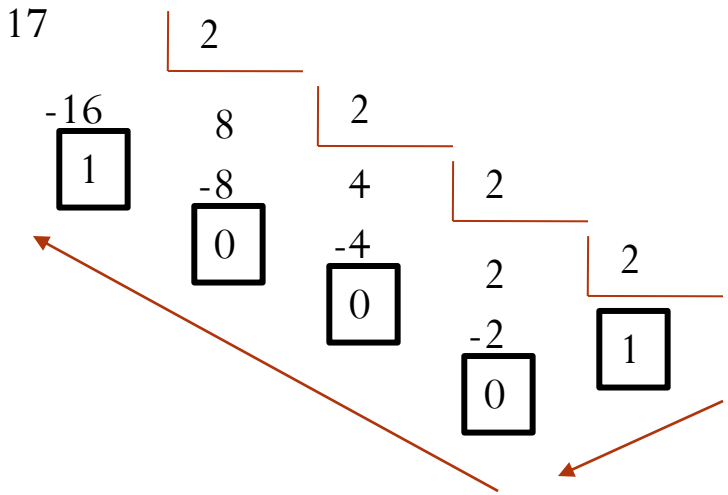
Tema 1

Representación de la información

Guillem-Jordi Esteve Vizcarra

Transformar un número decimal a binario



Resultado: 10001 es el número 17 en binario

Suma en binario

$$\begin{array}{r} 1001 \\ +\ 100 \\ \hline 1101 \end{array}$$

+1+1+1+1

$$\begin{array}{r} 1011 \\ +\ 111 \\ \hline 10010 \end{array}$$

ATENCIÓN:

0+0=0
0+1=1
1+0=1
1+1=0 y llevo 1
1+1+1 (una que llevo)= 1 y llevo 1

Resta en binario

$$\begin{array}{r} 1001 \\ {}^{+1}\,100 \\ \hline 0101 \end{array} \qquad \begin{array}{r} 1011 \\ {}^{+1}\,111 \\ \hline 0100 \end{array}$$

ATENCIÓN:

0-0=0
1-1=0
1-0=1
0-1=1 y llevo 1
1-1 (pero llevo 1)= 1 y llevo 1
0-1 (pero llevo 1)= 0 y llevo 1

$$\begin{array}{r} 1001 \\ {}^{+1+1}\,111 \\ \hline 0010 \end{array} \qquad \begin{array}{r} 1010 \\ {}^{+1+1+1}\,111 \\ \hline 0011 \end{array}$$

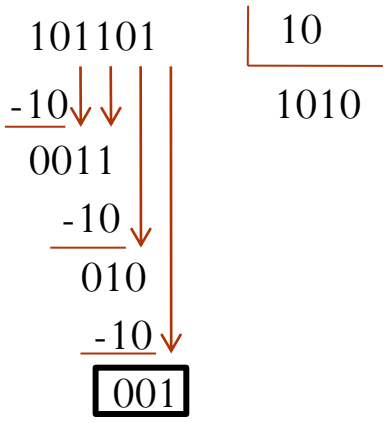
Multiplicación en binario

$$
\begin{array}{r}
1001 \\
\times \quad 11 \\
\hline
1001 \\
{\scriptstyle +1 \quad +1} \\
+1001 \\
\hline
10010
\end{array}
$$

ATENCIÓN:

0x0=0
1x1=1
1x0=0
0x1=0

División de un número en binario



1. Cogemos la parte divisible del dividendo, en este caso 10.
2. Multiplicamos el divisor por 1 y restamos, obteniendo el 00.
3. Bajamos la siguiente cifra. Como 1 no es divisible entre 10, bajaremos la siguiente y colocaremos un 0 en el cociente
4. Multiplicaremos el divisor por 1 y restaremos, dando la resta como resultado = 0.
5. Bajamos la siguiente cifra y como 1 no es divisible entre 10, obtenemos que el cociente = 1010 y el resto = 1

Teorema fundamental de la numeración.

Σ=Suma de elementos

d=Dígito

b=Base

p=Posición

$$\sum d*b^{p}$$

La base es la cantidad de dígitos de un sistema de numeración

En binario {0,1} la base=2
En decimal {0,1,..,9} base =10

Ejemplo:

Tenemos el número en binario 1011

Aplicamos la formula:

1*2^3+0*2^2+1*2^1+1*2^0=8+0+2+1=11

Por lo tanto el número binario 1011 = al número decimal 11

Cambios de Base

- Hasta ahora sabemos cambiar de binario a decimal y de decimal a binario.

- Para cambiar de decimal a cualquier base solo tenemos que dividir entre la base que necesitemos.

- Por ejemplo, para cambiar de decimal a binario dividíamos entre 2. Pues para cambiar de decimal a base 7 dividiremos entre 7.

Cambios de Base

- Si por alguna razón quisiéramos cambiar un número en base 5 a un número en base 7 el proceso sería el siguiente:
  - 1. Pasar el número en base 5 a base decimal.
  - 2. Pasar el número obtenido en base decimal a base 7.

Ejemplo:

32 (5 a Base 7

1. Aplicamos el teorema fundamental de la Numeración

3*5^1 + 2*5^0 = 15 + 2 = 17 (10

2. Dividimos entre 7

17/7=2 con resto 3

Por lo tanto el número 32 en Base (5 se convertirá en el 23 en Base (7

Signo-Magnitud

- Octeto / Byte = conjunto de 8 bits

- Palabra = Conjunto de bits. Estará en las diferentes potencias de 2 (2^2=4; 2^3=8; 2^4=16; 2^5=32)

- El signo magnitud dedica uno de sus bits a representar el signo de un número:
  - 0 = +
  - 1 = -

Signo-Magnitud

- Ejemplo:
- Se nos da el número 32 y lo queremos representar en signo magnitud, conociendo el ancho de palabra (n=8)

- Sabemos que vamos a dedicar 1 bit al signo (que será 0) y 7 al número en binario.

- Por lo tanto el resultado será:

0 (signo)	0100000 (número)

- Como vemos que el número 32 (100000) no ocupa los 7 bits que debería, rellenamos con 0's a su izquierda.

Complemento a 1

- Igual que en signo-magnitud: usa el bit de la izquierda para representar el signo.
- Nº's positivos: Igual que el signo-magnitud
- Nº's negativos: Complementando el nº positivo correspondiente (cambiando 0 por 1 y viceversa)

Complemento a 1

- Ejemplos:
  - -20 → 20 = 00010100 → 11101011
  - 73 → 01001001
  - -12 → 12 = 00001100 → 11110011

Complemento a 1

- Inconveniente: 2 posibles representaciones para el 0
- +0 → 00000000 Ca1 → 11111111

Complemento a 2

- Igual que los anteriores: usa el bit de la izquierda para representar el signo.
- Nº's positivos: Igual que el signo magnitud
- Nº's negativos: Se realiza el Ca1 (cambiar el 0 por 1, incluido el signo) y se le suma 1 en binario, despreciando el acarreo si existe

Complemento a 2

- Ejemplo:

-10 → 10 = 00001010 → 11110101 (Ca1) → 11110110 (Ca2)

- Ventaja: Una única representación para el 0

Representación en exceso a 2^{n-1}

- No usa bit para el signo. Todos los bits representan un valor = nº + exceso (para n bits)

- Con ello, el nº resultante será positivo y se representará en binario natural.

Representación en exceso a 2^{n-1}

- Ejemplo:

Si n=8 bits → Exceso = $2^{8-1} = 2^7 = 128$ → Cuando nos den un nº a representar, le sumaremos 128

10 → 10 + 128 = 138 → 10001010

-10 → -10 + 128 = 118 → 01110110

109 → 109 + 128 = 237 → 11101101

-109 → -109+128 = 19 → 00010011

Representación en exceso a 2^{n-1}

- Ejemplo:

127 → 127+128 = 255 → 11111111

-127 → -127+128 = 1 → 00000001

128 → 128+128 = 256 →100000000 (9 bits, excede la cantidad de 8 que nos dan → no se puede representar)

-128 → -128 +128 = 0 → 00000000

Suma en Complemento a 1

- Nº's positivos: En Ca1 sumamos igual que en binario natural, y si existe acarreo en el bit de más a la izquierda, lo sumamos al resultado.

- Nº's negativos: Convertimos el sustraendo a Ca1 y lo sumamos al minuendo: $a - b = a + (-b)$

Suma en Complemento a 1

- Ejemplo: 12-5 = 12 + (-5) → n=8bits

12 = 00001100

-5 en Ca1 → 5 = 00000101 → 11111010 = -5

```
  00001100
+ 11111010
----------
 100000110
+        1
----------
  00000111
```

Suma en Complemento a 1

- Ejemplo: 5 - 12 = 5 + (-12) → n=8bits

5 = 00000101

-12 en Ca1 → 12 = 00001100 → 11110011 = -12

```
  00000101
+11110011
 - 11111000
```

Como está en Ca1 y tenemos signo negativos: complementamos a la inversa: 00000111 = 7 → -7

Suma en Complemento a 2

- Nº's positivos: En Ca2 sumamos igual que en binario natural, y si existe acarreo en el bit de más a la izquierda, lo desechamos.

- Nº's negativos: Convertimos el sustraendo a Ca2 y lo sumamos al minuendo: a – b = a + (-b)

Suma en Complemento a 2

- Ejemplo: 12-5 = 12 + (-5) → n=8bits

12 = 00001100

-5 en Ca1 → 5 = 00000101 → 11111010 = -5 (Ca1)

→ 11111011 (Ca2)

```
  00001100
+ 11111011
 100000111
```

Desechamos el exceso

Suma en Complemento a 2

- Ejemplo: 5 - 12 = 5 + (-12) → n=8bits

5 = 00000101

-12 en Ca2 → 12 = 00001100 → 11110011 = -12 (Ca1)

→ 11110100 (Ca2)

```
  00000101
+ 11110100
 - 11111001
```

Como está en Ca2 y tenemos signo negativo: restamos 1 →11111000

Como está en Ca1 y tenemos signo negativo: complementamos a la inversa: 00000111 = 7 → -7

Error de desbordamiento

- Al realizar sumas de nºs del mismo signo y al tener un ancho de palabra determinado, ocurrirá un error de desbordamiento, y el resultado aparecerá con el signo contrario al de los sumandos.

- Ejemplo: Con n= 8 bits, si sumamos 115 + 25

```
  01110011   (115)
+ 00011001    (25)
  10001100 = -12
```

Representación de la información

- Coma o punto fijo
- Coma o punto flotante
  - Simple precisión
  - Doble precisión
  - IEEE754
- Representación de datos alfabéticos y alfanuméricos
- EBCDIC
- ASCII
- Unicode

Representación de la información

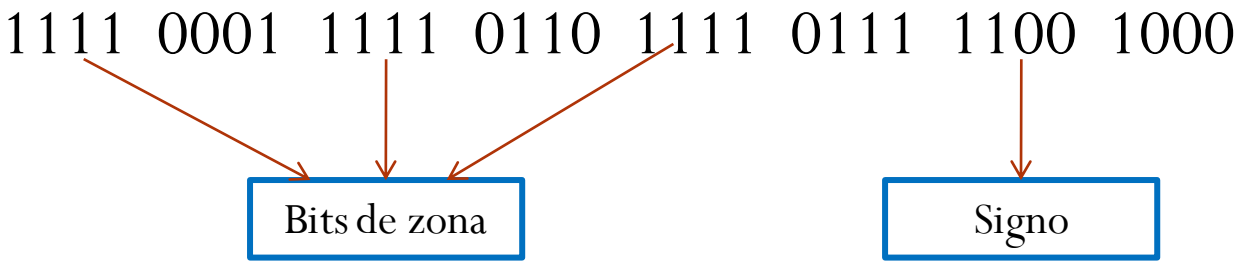
- Coma o punto fijo
  - Binario Puro (dado anteriormente)
  - Decimal desempaquetado
  - Decimal empaquetado

Coma o punto fijo – Decimal desempaquetado

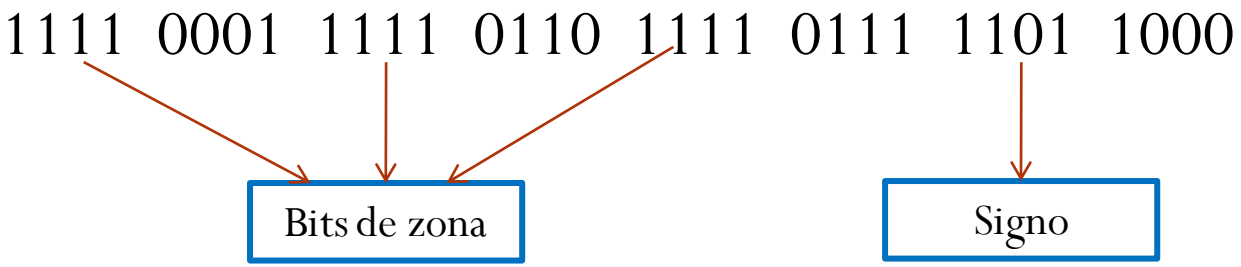
- Cada dígito ocupará 1 byte
  - Cuarteto izquierda: 1111 = F (bits de zona)
  - Cuarteto derecha: cifra codificada en binario (bits de dígito)
- En el último byte: el cuarteto de la izquierda representa el signo
  - 1100 = positivo = C
  - 1101 = negativo = D

Coma o punto fijo – Decimal desempaquetado

- Ej: 1678



- Ej: -1678



Coma o punto fijo – Decimal empaquetado

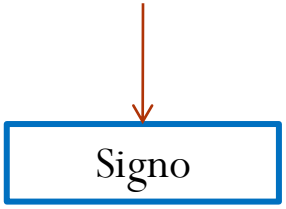
- Cada dígito ocupará 1 cuarteto (sin bits de zona

- El primer dígito de la derecha lleva a su derecha el signo
  - 1100 = positivo = C
  - 1101 = negativo = D

Coma o punto fijo – Decimal empaquetado

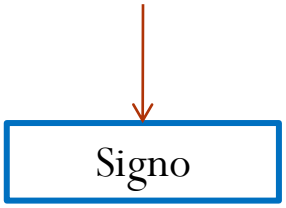
- Ej: 1678

0000 0001 0110 0111 1000 1100



- Ej: -1678

0000 0001 0110 0111 1000 1101



Coma o punto flotante

- Se utiliza la notación científica normalizada:
  - Número = Mantixa * $Base^{Exp}$
  - Mantisa: No tiene parte entera y el primer dígito decimal es significativo

- Ejemplos:
  - 123.45 → $0.12345 * 10^{3}$
  - 6789.1 → $0.67891 * 10^{4}$
  - 0.00056 → $0.56 * 10^{-3}$
  - $62.3 * 10^{4}$: no es normalizada → $0.623 * 10^{6}$
  - $0.02 * 10^{-3}$: no es normalizada → $0.2 * 10^{-4}$

Coma o punto flotante

- Valores que almacenará el ordenador para un nº en coma flotante:
  - Mantisa (+ signo): El lugar de la coma se supone a la izquierda de la mantisa y no se almacena.
  - Exponente (+signo)
  - El valor de la base va implícito es el sistema elegido (en los ejemplos anteriores: el 10; para el ordenador: binario (2)).

- La coma flotante se puede representar de 3 formas (en función del ancho de palabra usado)

Coma o punto flotante – Simple precisión

- Usando n bits:

Signo	Exponente	Mantisa
1 bit	m = exp.	Valor matisa = n-1-m (en Ca1)
0=+	exceso a 2^{m-1}	
1=–		

Coma o punto flotante – Simple precisión

- **Ejemplo: Representación del nº 12 con un ancho de palabra de 32 bits, con 8 para el exponente**
- n = 32 bits
- m = 8 bits (exp.) → exceso = $2^{8-1} = 128$
- mantisa = 32-1-8 = 23 bits
- **1º: Signo**: positivo → 0
- **2º: Exponente**. Debemos expresar 12 en notación normalizada en base 2.

Coma o punto flotante – Simple precisión

- En general:
- $\text{cantidad}/\text{base}^{\text{nº de dígitos necesarios para representar la cantidad en esa base}}$
- $12 < 2^4 = 16$
- Exponente = exceso 4 + 128 = 132 → 10000100
- 12/16 = 0.75
- **3º: Mantisa:** 0.75
  - 0.75 * 2 = 1.5 → 1
  - 0.5 * 2 = 1 → 1 (se acaba y se rellena de 0s)

0 | 1000100 | 11000000000000000000000

IEEE754

- Usando n bits:

Signo	Exponente	Mantisa
1 bit	m = exp.	Valor matisa = n-1-m (en Ca1)
0=+	exceso a 2^{m-1}	
1=–		

IEEE754

- Utiliza notación exponencial → $N^o = Mantisa * Base^{Exp}$
- Se reserva un bit para el signo
- El exponente se codifica en exceso $2^{m-1}-1$. Donde m es el número de bits reservados para el exponente
- El valor guardado en el exponente será el exponente más el exceso.
- La mantisa se almacena generalmente en posición normalizada con el primer 1 implícito.
- Para una palabra de n = 32 bits, y m = 8 de exponente, el tamaño de la mantisa será n-m-1 = 32-8-1 = 23 bits más 1 implícito.

IEEE754

- **Ejemplo: Representación del nº 17.5 con un ancho de palabra de 32 bits, con 8 para el exponente**
- n = 32 bits
- m = 8 bits (exp.) → exceso = $2^{8-1}-1 = 127$
- mantisa = 32-1-8 = 23 bits
- **1º: Signo**: positivo → 0
- **2º: Exponente**. Debemos expresar 12 en notación normalizada en base 2.

IEEE754

- En general:
- cantidad/base$^{\text{nº de dígitos necesarios para representar la cantidad en esa base}}$
- $17.5 < 2^5 = 32$
- $17.5 < 2^4 = 16$
- Exponente = exceso 4 + 127 = 131 → 10000011
- 17.5/16 = 1.09375
- 1.09375 = 1.00011 Recuerda que luego en la representación el 1 va implícito, no se almacenará, solo se almacenará el 00011

IEEE754

- **3º: Mantisa:** 1.09375
  - 0.09375 * 2 = 0.1875 → 0
  - 0.1875 * 2 = 0.375 → 0
  - 0.375 * 2 = 0.75 → 0
  - 0.75 * 2 = 1.5 → 1
  - 0.5 * 2 = 1 → 1 (Se acaba y se rellena de 0s)

0 | 1000011 | 00011000000000000000000

Representación de datos alfabéticos y alfanuméricos

- Hasta ahora hemos tratado cantidades numéricas; ahora veremos como representar el alfabeto.
- La estandarización ha llevado a los códigos de entrada/salida:
  - BCD
  - EBCDIC
  - ASCII
  - Unicode

BCD

- Codifica los símbolos numéricos del 0 al 9. Divide cada byte en 2 cuartetos y en cada uno almacena en binario una cifra:

Decimal	Binario	Decimal	Binario
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

- Ej: 148 → BCD: 0000 0001 0100 1000

EBCDIC

- BCD extendido para intercambio de información. Usado en los PC IBM. Cada carácter tiene 8 bits → Podremos representar $28 = 256$ caracteres.

- Almacena mayúsculas / minúsculas / caracteres especiales / caracteres de control para dispositivos de entrada/salida y para comunicaciones

- Cada carácter se divide en dos partes:
  - 4 bits de zona
  - 4 bits de dígito

EBCDIC

- Los bits de zona indican el tipo de carácter:
  - Números: bits de zona = 1111 (F)
  - A - I = 1100 (C)
  - J - R = 1101 (D)
  - S - Z = 1110 (E)
  - a – i = 1000 (8)
  - j – r = 1001 (9)
  - s – z = 1010 (A)
  - Caracteres especiales: 01xx
  - Caracteres sin asignar: 00xx

ASCII

- Usado por S.O.: MS-DOS, Windows, Unix
- Con 7 bits por carácter → $2^7 = 128$ caracteres (letras mayúsculas, minúsculas y otros símbolos)
- Extendido: 8 bits (para más símbolos de otros lenguajes y símbolos gráficos

Unicode

- Usado por Win NT, Internet explorer, Netscape
- Con 16 bits da un nº único a cada carácter independientemente del idioma.
- Se incorpora su uso a sitios web y aplicaciones cliente-servidor.