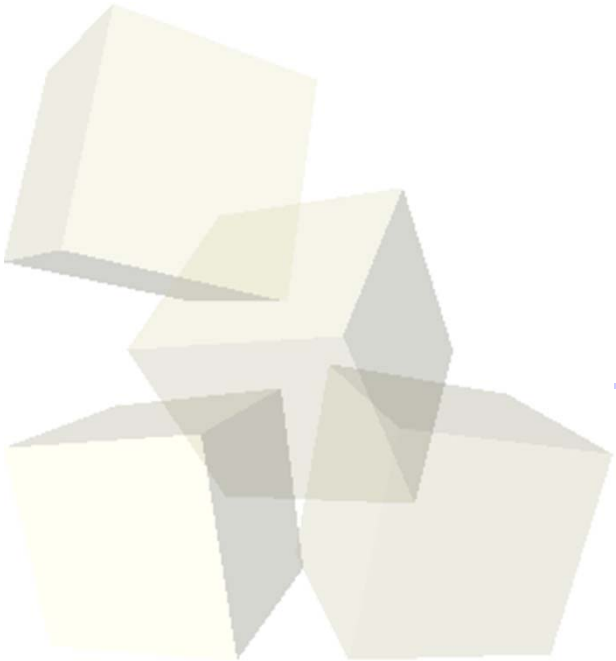


Tema 3

Aritmética y representación de la información en el ordenador

Informática
Grado en Física
Universitat de València

Francisco.Grimaldo@uv.es
Ariadna.Fuertes@uv.es

Guión del tema

- Representación binaria de la información
  - Definición
  - Agrupación de bits
  - Unidades de medida binarias
- Información básica a representar
  - Información booleana
  - Conversión decimal – binario
  - Operaciones en binario
  - Enteros con signo
  - Caracteres
  - Reales
- Representación intermedia: octal y hexadecimal
  - Definición
  - Conversión decimal – hexadecimal/octal - binario
- Ejercicios

Representación binaria: definición (1/2)

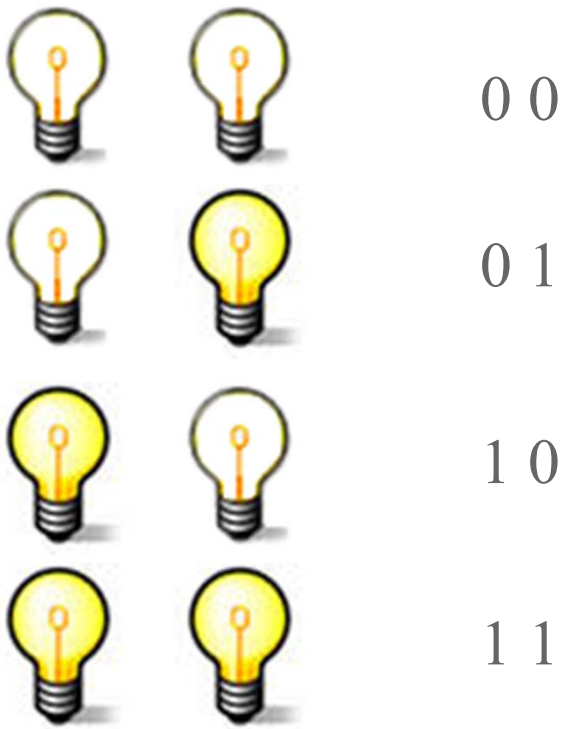
- Los ordenadores representan cualquier tipo de información (texto, imágenes, ...) como un patrón de bits que pueden estar en dos estados posibles: encendido (1) o apagado (0).
- El bit es la unidad mínima de información que puede procesar un dispositivo digital.
- Un bit tiene uno de los dos valores posibles: 0 ó 1.
- A pesar de esto, un ordenador trabaja con información diferente de ceros y unos…. ¿cómo lo hace?... Codificando…

Representación binaria: definición (2/2)

- Con un bit no podemos más que representar dos valores, 0 y 1.
- Para poder representar o codificar más información en un dispositivo digital, necesitamos una cantidad mayor de bits.
- Por medio de secuencias de bits podemos representar cualquier tipo de información.
- … esto se conoce como codificación

Agrupación de bits

- Por ejemplo, si disponemos de dos bits podremos representar hasta cuatro valores diferentes, 2^2
- En general, teniendo en cuenta que cada bit solo puede representar dos valores (0 y 1), con n bits podremos representar 2^n valores.
- Byte es una agrupación de 8 bits.
- Un byte puede representar hasta 256 (2^8) eventos diferentes. Ej: 10010011.



Unidades de medida binarias

- Byte: grupo de 8 bits.
- KB (kilobyte o K): grupo de 1.024 bytes. Contiene 2^{10} bytes = 1.024 bytes ≈ 1.000 bytes = 10^3.
- MB (megabyte o mega): grupo de 1.048.576 bytes Contiene 2^{20} bytes ≈ 1.000.000 bytes = 1.000 KB = 10^6.
- GB (gigabyte o giga): 1.024 MB ≈ 1000 MB.
- TB (terabyte): 1 millón de MB o billón de bytes.
- PB (petabyte): 1.024 TB o 1000 billones de bytes.

- Ejemplos: 4 GB RAM, HDD 120 GB, MP3 3,5 MB.
- La transferencia de datos se mide en bits/s. Por tanto, 1 Mb/s = 1.000 bits = 1/8 MB

Guión del tema

- Representación binaria de la información
  - Definición
  - Agrupación de bits
  - Unidades de medida binarias
- Información básica a representar
  - Información booleana
  - Conversión decimal – binario
  - Operaciones en binario
  - Enteros con signo
  - Caracteres
  - Reales
- Representación intermedia: octal y hexadecimal
  - Definición
  - Conversión decimal – hexadecimal/octal - binario
- Ejercicios

Información básica a representar

- La información básica a representar/almacenar en un dispositivo digital va a ser:
  - Información booleana: Verdadero o Falso.
  - Valores enteros: Números enteros positivos y negativos.
  - Caracteres: letras del abecedario de la “a” a la “z ”, en mayúsculas y minúsculas, los dígitos numéricos del 0 al 9 y otros caracteres de control.
  - Valores reales: Números con decimales.
- Toda esta información debe estar codificada en binario para poderse almacenar en el dispositivo digital.

- Codificación de la información booleana: Verdadero = 1 y Falso = 0

Conversión decimal – binario (1/4)

- Un número entero positivo cualquiera, representado en base decimal, se puede codificar como una suma de potencias de 10. Por ejemplo: $325 = 3*10^2 + 2*10^1 + 5*10^0$.

- En general, un número **N** se puede codificar partiendo de un conjunto ***a*** de símbolos (alfabeto) que toman un determinado valor en función de la posición que ocupan B^i (base del sistema de numeración o representación) de la manera siguiente:

$$N = \sum_{i=0}^{n} a_i * B^i$$

Conversión decimal – binario (2/4)

- La base con que trabajamos habitualmente es base 10 (B=10) y el alfabeto con que escribimos normalmente los números son los diez dígitos {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.

- El número 237 se puede codificar mediante su base y los coeficientes correspondiente: $2*10^2+3*10^1+7*10^0$, es decir, 2*100+3*10+7*1=237.

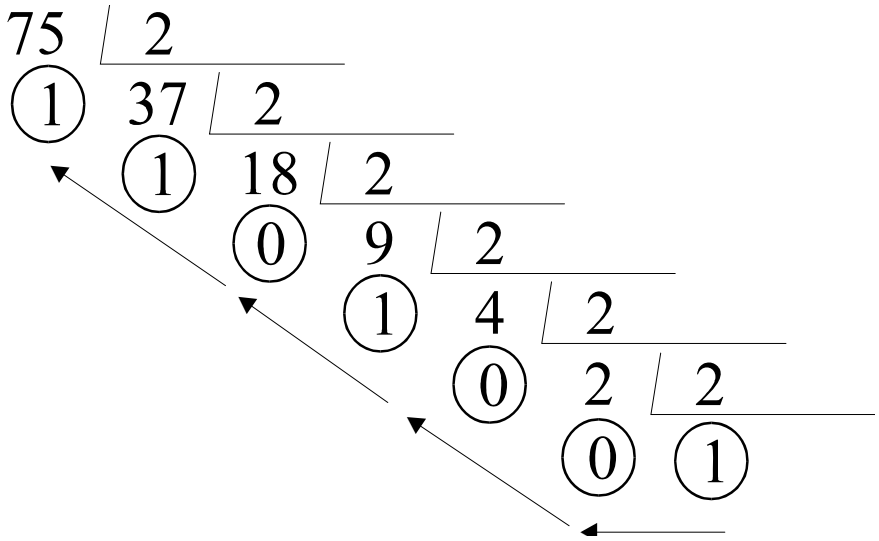
- Los ordenadores usan un alfabeto formado por los dígitos 0 y 1 (dos símbolos), de manera que la base que utilizan es la base binaria (B=2).

- ¿Cómo convertimos un número en base 10 en un número en base 2 o binario?

Conversión decimal – binario (3/4)

- El cambio de base de decimal a binario se hace mediante divisiones sucesivas.

- Por ejemplo, para pasar 75 a binario hacemos la siguiente secuencia de divisiones y, así $75_{)10} = 1001011_{)2}$



- Cuando ya no podemos dividir más el número, **la sucesión inversa desde el último cociente más los restos de las divisiones son lo que nos forman el número binario.**

Conversión decimal – binario (4/4)

- Para pasar de binario a decimal, haremos la descomposición de la cifra en las sucesivas potencias según la fórmula (con B=2)

$$N=\sum_{i=0}^{n} a_i * 2^i$$

- Por ejemplo, para convertir el número 1001011 a base 10, lo descompondremos de la siguiente manera:

1001011 = $1*2^0+1*2^1+0*2^2+1*2^3+0*2^4+0*2^5+1*2^6$
= $1*1+1*2+0*4+1*8+0*16+0*32+1*64$
= $1+2+8+64$
= 75

Operaciones en binario

- La metodología de las operaciones es la misma que en decimal (En binario: 1+1 = 10 ó 10-1=01)

- **Ejemplo**: 75 + 23 (en decimal 98 ≡ 1100010)
  75 ≡ 1001011
  23 ≡ 10111

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

El resultado es el esperado:

$0*2^0+1*2^1+0*2^2+0*2^3+0*2^4+1*2^5+1*2^6 = 98$

- **Ejemplo**: 75 - 23 (en decimal 52 ≡ 110100)

```
  1 0 0 1 0 1 1
-     1 0 1 1 1
-------------------
  0 1 1 0 1 0 0
```

El resultado es el esperado:

$0*2^0+0*2^1+1*2^2+0*2^3+1*2^4+1*2^5+0*2^6 = 52$

Enteros con signo (1/9)

- **Representación signo - magnitud:** consiste en dedicar un espacio para indicar el signo del número.
  - El número 75 puede ser +75 o -75.

- En la memoria del ordenador es imposible representar los símbolos + ó -, sólo se pueden representar unos y ceros.

- Lo que se hace es dedicar un espacio (bit) concreto de la representación al signo, que también será cero o un uno, pero que por estar en una cierta posición no indicará magnitud sino signo.

- Si hablamos de un lugar determinado dónde irá el signo, también tendremos que hablar de un número determinado de espacios para poder localizar el signo y la magnitud. Así, **de ahora en adelante, cualquier representación binaria de un número llevará pareja el número de bits en que se realizará la representación** (y a su vez, el método en que ha sido representado).

Enteros con signo (2/9)

- Ejemplo: Representar los números $75_{)10}$ y $-75_{)10}$ en base binaria, en signo magnitud en 8 bits (n=8)

$$75_{)10} = 01001011_{)2SM}$$
$$-75_{)10} = 11001011_{)2SM}$$

- Si al realizar la representación apareciesen huecos (casillas sin rellenar) éstas se han de dejar siempre a la derecha del signo y a la izquierda de la magnitud, y se rellenarán siempre con ceros:

$$8_{)10} = 0\underline{000}1000_{)2SM}$$
$$-8_{)10} = 1\underline{000}1000_{)2SM}$$

- **Inconveniente** de esta representación: a la hora de realizar operaciones (se debe comprobar el signo y dependiendo de este realizar la suma o la resta, ver que magnitud es mayor,…).

- **Ejercicio**: convertir el número $-46_{)10}$ a binario con 8 bits SM.

Enteros con signo (3/9)

- **Representación complemento a 1:** consiste en aplicar a un numero entero ***x*** la siguiente trasformación:

$$x' = (2^n - 1) + x$$

teniendo en cuenta el "acarreo" en las operaciones.

- **Ejemplo**: Representar todos los números posibles en 4 bits (n=4)

$2^4 = 10000 \rightarrow 2^4 - 1 = 1111$

$0 \rightarrow 1111 + 0 = 1111)_{2C1}$
$1 \rightarrow 1111 + 1 = (1)0000 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0000 + 1 = 0001)_{2C1}$
$2 \rightarrow 1111 + 10 = (1)0001 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0001 + 1 = 0010)_{2C1}$
$3 \rightarrow 1111 + 11 = (1)0010 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0010 + 1 = 0011)_{2C1}$
$4 \rightarrow 1111 + 100 = (1)0011 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0011 + 1 = 0100)_{2C1}$
$5 \rightarrow 1111 + 101 = (1)0100 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0100 + 1 = 0101)_{2C1}$
$6 \rightarrow 1111 + 110 = (1)0101 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0101 + 1 = 0110)_{2C1}$
$7 \rightarrow 1111 + 111 = (1)0110 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0110 + 1 = 0111)_{2C1}$
$8 \rightarrow 1111 + 1000 = (1)0111 \rightarrow$ (se tiene en cuenta el acarreo) $\rightarrow 0111 + 1 = 1000)_{2C1}$
(número no válido porque es igual que el –7)

$-1 \rightarrow 1111 - 1 = 1110)_{2C1}$
$-2 \rightarrow 1111 - 10 = 1101)_{2C1}$
$-3 \rightarrow 1111 - 11 = 1100)_{2C1}$
$-4 \rightarrow 1111 - 100 = 1011)_{2C1}$
$-5 \rightarrow 1111 - 101 = 1010)_{2C1}$
$-6 \rightarrow 1111 - 110 = 1001)_{2C1}$
$-7 \rightarrow 1111 - 111 = 1000)_{2C1}$
$-8 \rightarrow 1111 - 1000 = 0111)_{2C1}$ (número no válido porque es igual que el 7)

Enteros con signo (4/9)

- Características del C1:
  - El 1er bit de los núm. positivos es un 0 y de los negativos es un 1.
  - Los números positivos se representan igual pero completados con ceros a la izquierda hasta el número total de bits de la representación.
  - Los números negativos se pueden obtener aplicando la fórmula o bien siguiendo los siguientes pasos:
    1. Representar el número en binario como si fuese positivo.
    2. Completar con ceros hasta el número de bits de representación
    3. Cambiar los ceros por unos y unos por ceros.
  - La suma y la resta se convierte en una única operación: SUMA

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

- Cualquier representación binaria lleva fijado el tamaño. Si el tamaño elegido no es bueno (es pequeño), podemos encontrarnos con dos tipos de errores al realizar operaciones:
  - **Overflow**: Se debería obtener un núm. positivo y sale negativo
  - **Underflow**: cuando debería ser negativo y sale positivo.

Enteros con signo (5/9)

- **Ejemplo 1** : Supongamos n = 4. Sumar A=5 y B=2

  $A = 5_{)10} = 0\ 1\ 0\ 1)_{2C1}$

  $B = 2_{)10} = 0\ 0\ 1\ 0)_{2C1}$

  $A+B = 0\ 1\ 1\ 1)_{2C1} = 7_{)10}$

```
  0 1 0 1
+ 0 0 1 0
  0 1 1 1
```

- **Ejemplo 2** : Supongamos n = 4. Restar A=5 y B=6 ≡ A + (-B)

  $A = 5_{)10} = 0\ 1\ 0\ 1)_{2C1}$

  $-B = -6_{)10} = -\ 0\ 1\ 1\ 0)_2 = 1\ 0\ 0\ 1)_{2C1}$

  $A+(-B) = 1\ 1\ 1\ 0)_{2C1} = -\ 0\ 0\ 0\ 1)_2 = -1_{)10}$

```
  0 1 0 1
+ 1 0 0 1
  1 1 1 0
```

- **Ejemplo 3** : Supongamos n = 4. Sumar A=5 y B=6

  $A = 5_{)10} = 0\ 1\ 0\ 1)_{2C1}$

  $B = 6_{)10} = 0\ 1\ 1\ 0)_{2C1}$

  $A+B = 1\ 0\ 1\ 1)_{2C1} = -\ 0\ 1\ 0\ 0)_2 = -4_{)10} \rightarrow$ Overflow

```
  0 1 0 1
+ 0 1 1 0
  1 0 1 1
```

- **Ejemplo 4** : Supongamos n = 4. Sumar A=-5 y B=-6

  $A = -5_{)10} = -\ 0\ 1\ 0\ 1)_2 = 1\ 0\ 1\ 0)_{2C1}$

  $B = -6_{)10} = -\ 0\ 1\ 1\ 0)_2 = 1\ 0\ 0\ 1)_{2C1}$

  $A+B = 0\ 1\ 0\ 0)_{2C1} = 4_{)10} \rightarrow$ Underflow

```
  1 0 1 0
+ 1 0 0 1
(1) 0 0 1 1
```

El acarreo se suma:
0011+1=0100

Enteros con signo (6/9)

- **Pequeño inconveniente** de esta representación: el cero tiene dos representaciones de manera que tenemos representados 'sólo' 2^n-1 diferentes números, cuando podríamos tener representados 2^n.

- **Ejercicio(*)**: convertir los siguientes números a binario con 8 bits C1 y realizar las operaciones verificando si el resultado es correcto. Si no lo es, detectar el tipo de error que se produce.

  $A = 45_{)10}$
  $B = -86_{)10}$

  A + B

  B – A

  A – B

Enteros con signo (7/9)

- **Representación complemento a 2:** consiste en aplicar a un numero entero ***x*** la misma trasformación que en C1 pero sumándole 1 y no teniendo en cuenta luego el acarreo:

$$x' = (2^n-1)+x+1$$

- **Ejemplo**: Representar todos los números posibles en 4 bits (n=4)

$2^4 = 10000 \rightarrow 2^4 - 1 = 1111$

$0 \rightarrow 1111 + 0 + 1 = (1)\ 0\ 0\ 0\ 0 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow \quad 0\ 0\ 0\ 0)_{2C2}$

$1 \rightarrow 1111 + 1 + 1 = (1)\ 0\ 0\ 0\ 1 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow \quad 0\ 0\ 0\ 1)_{2C2}$

$2 \rightarrow 1111 + 10 + 1 = (1)\ 0\ 0\ 1\ 0 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow \quad 0\ 0\ 1\ 0)_{2C2}$

$3 \rightarrow 1111 + 11 + 1 = (1)\ 0\ 0\ 1\ 1 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow \quad 0\ 0\ 1\ 1)_{2C2}$

$4 \rightarrow 1111 + 100 + 1 = (1)\ 0\ 1\ 0\ 0 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow 0\ 1\ 0\ 0)_{2C2}$

$5 \rightarrow 1111 + 101 + 1 = (1)\ 0\ 1\ 0\ 0 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow 0\ 1\ 0\ 1)_{2C2}$

$6 \rightarrow 1111 + 110 + 1 = (1)\ 0\ 1\ 1\ 0 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow 0\ 1\ 1\ 0)_{2C2}$

$7 \rightarrow 1111 + 111 + 1 = (1)\ 0\ 1\ 1\ 1 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow 0\ 1\ 1\ 1)_{2C2}$

$8 \rightarrow 1111 + 1000 + 1 = (1)\ 1\ 0\ 0\ 0 \rightarrow$ y NO se tiene en cuenta el acarreo $\rightarrow 1\ 0\ 0\ 0)_{2C2}$

(número no válido porque es igual que el –8)

$-1 \rightarrow 1111 - 1 + 1 = 1\ 1\ 1\ 1)_{2C2}$

$-2 \rightarrow 1111 - 10 + 1 = 1\ 1\ 1\ 0)_{2C2}$

$-3 \rightarrow 1111 - 11 + 1 = 1\ 1\ 0\ 1)_{2C2}$

$-4 \rightarrow 1111 - 100 + 1 = 1\ 1\ 0\ 0)_{2C2}$

$-5 \rightarrow 1111 - 101 + 1 = 1\ 0\ 1\ 1)_{2C2}$

$-6 \rightarrow 1111 - 110 + 1 = 1\ 0\ 1\ 0)_{2C2}$

$-7 \rightarrow 1111 - 111 + 1 = 1\ 0\ 0\ 1)_{2C2}$

$-8 \rightarrow 1111 - 1000 + 1 = 1\ 0\ 0\ 0)_{2C2}$

$-9 \rightarrow 1111 - 1001 + 1 = 0\ 1\ 1\ 1)_{2C2}$ (número no válido porque es igual que el 7)

Enteros con signo (8/9)

- Características del C2:
  - El 1er bit de los núm. positivos es un 0 y de los negativos es un 1.
  - Los números positivos se representan igual pero completados con ceros a la izquierda hasta el número total de bits de la representación.
  - Los números negativos se pueden obtener aplicando la fórmula o bien siguiendo los siguientes pasos:
    1. Representar el número en binario como si fuese positivo.
    2. Completar con ceros hasta el número de bits de representación
    3. Cambiar los ceros por unos y unos por ceros
    4. Sumarle 1 al resultado
  - La suma y la resta se convierte en una única operación: SUMA

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

El C2 es “casi igual” que C1, sólo hay que sumarle 1 y sin acarreo.

Ejercicio(*):Realizar el mismo ejercicio que en C1 pero ahora en C2

Enteros con signo (9/9)

- **Ejemplo 1** : Supongamos n = 4. Sumar A=5 y B=2

  $A = 5_{)10} = 0\ 1\ 0\ 1)_{2C2}$

  $B = 2_{)10} = 0\ 0\ 1\ 0)_{2C2}$

  $A+B = \mathbf{0\ 1\ 1\ 1)_{2C2}} = 7_{)10}$

```
  0 1 0 1
+ 0 0 1 0
  0 1 1 1
```

- **Ejemplo 2** : Supongamos n = 4. Restar A=5 y B=6 ≡ A + (-B)

  $A = 5_{)10} = 0\ 1\ 0\ 1)_{2C2}$

  $-B = -6_{)10} = -\ 0\ 1\ 1\ 0)_2 = 1\ 0\ 1\ 0)_{2C2}$

  $A+(-B) = \mathbf{1\ 1\ 1\ 1)_{2C2}} = 1\ 1\ 1\ 0)_{2C1} = -\ 0\ 0\ 0\ 1)_2 = -1_{)10}$

```
  0 1 0 1
+ 1 0 1 0
  1 1 1 1
```

- **Ejemplo 3** : Supongamos n = 4. Sumar A=5 y B=6

  $A = 5_{)10} = 0\ 1\ 0\ 1)_{2C2}$

  $B = 6_{)10} = 0\ 1\ 1\ 0)_{2C2}$

  $A+B = \mathbf{1\ 0\ 1\ 1)_{2C2}} = 1\ 0\ 1\ 0)_{2C1} = -\ 0\ 1\ 0\ 1)_2 = -5_{)10} \rightarrow$ Overflow

```
  0 1 0 1
+ 0 1 1 0
  1 0 1 1
```

- **Ejemplo 4** : Supongamos n = 4. Sumar A=-5 y B=-6

  $A = -5_{)10} = -\ 0\ 1\ 0\ 1)_2 = 1\ 0\ 1\ 1)_{2C2}$

  $B = -6_{)10} = -\ 0\ 1\ 1\ 0)_2 = 1\ 0\ 1\ 0)_{2C2}$

  $A+B = 0\ 1\ 0\ 1)_{2C2} = 5_{)10} \rightarrow$ Underflow

```
  1 0 1 1
+ 1 0 1 0
(1) 0 1 0 1
```

El acarreo se ignora!

Caracteres (1/5)

- La representación de caracteres alfanuméricos y numéricos se hace mediante una cierta correspondencia entre los caracteres que queremos representar y una serie de números binarios.
- La correspondencia más conocida y usada es el código ASCII *(American Standard Code for Information Interchange)* originalmente utilizado para comunicar información entre distintas máquinas.
- Además de los caracteres normales utilizados en la escritura, puede representar una serie de caracteres especiales con un cierto significado en los ordenadores, también conocidos como caracteres de control (Ej.: *enter*, *escape*, etc.).
- ASCII utiliza un *byte* (8 bits) para representar cada carácter.

Caracteres (2/5)

- El código ASCII original utilizaba siete bits para representar la información, y utilizaba el octavo bit (bit de control de errores) para comprobar la corrección de los otros siete controlando la paridad de los bits (número par o impar de unos del número binario).
  - Ejemplo: el número binario 0110101 tiene un número par de unos (4), por tanto, el bit de control será 0; mientras que el número 0100101 tendrá como bit de control un 1.
- Con los siete bits se puede llegar a representarse hasta un máximo de $2^7 = 128$ caracteres.
- Actualmente, gracias a la alta difusión del código ASCII para la representación de caracteres, se han introducido nuevos caracteres internacionales: vocales acentuadas, la ‘ñ', la ‘ç', etc. Esta **extensión del código ASCII** emplea el octavo bit y es específica de cada país.

Caracteres (3/5)

Dec	Hex	Car	Dec	Hex	Car	Dec	Hex	Car	Dec	Hex	Car
0	00	NUL	32	20	SPC	64	40	@	96	60	‘
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	’	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Código ASCII estándar

Caracteres (4/5)

- En el año 1991 aparece el estándar UNICODE.
- Unicode pretende codificar simultáneamente los caracteres de múltiples idiomas.
- Unicode provee de una codificación única para referirse a cada carácter de cada lengua cubierta.
- Actualmente, cubre los sistemas de símbolos de muchas lenguas, como la árabe, la birmana o la hebrea.
- ¡Soporta más de 100.000 caracteres únicos!

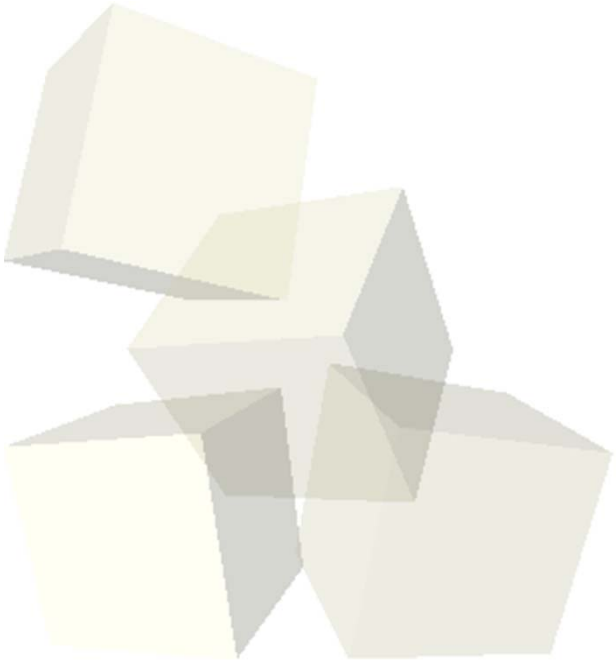
Caracteres (5/5)

- UNICODE deja la tarea de representación a la aplicación que ha de mostrar los caracteres (Ej.: navegador web).
- Unicode nada más asigna un código único a cada símbolo de cada lengua.
- Hay un método de asignación entre los códigos y los símbolos, una norma de transición.
- Existen diversos estándares de mapeo entre los códigos y los símbolos. Por ejemplo:
  - UTF-8, UTF-16, UTF-32
  - UCF-2, UCF-4

Reales (1/11)

- Para la representación de números reales tendremos que trabajar con una cierta precisión (no se pueden representar un número infinito de decimales).

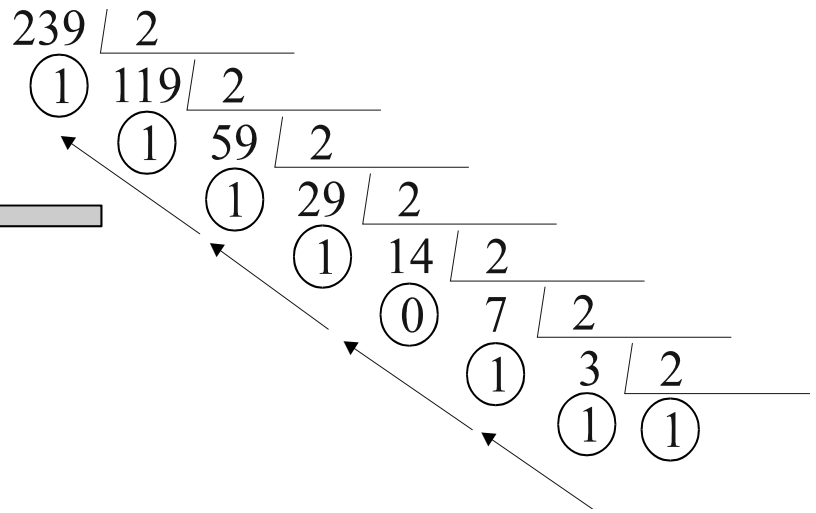
- En general, se puede representar de dos formas:
  - Mediante coma fija
  - Mediante coma flotante (o notación científica)



- **Representación en coma fija:**
  - Para pasar un número real de decimal a binario se ha de codificar, por un lado, la parte entera y, por otro, la parte decimal:
    - 239,403 → 239 y 403.
  - La parte entera se convierte como hemos visto antes con los números enteros: mediante divisiones sucesivas.
  - La parte decimal se convierte mediante multiplicaciones sucesivas, de las que ahora debemos coger la parte entera resultante de cada multiplicación.
  - **Importante:** Un real (ej.: 239,403) con número finito de decimales (403 son 3 decimales), puede no tener un número finito de decimales en su representación binaria!

- **Representación en coma fija (continuación):**

- Ejemplo: 239,403 a binario.
  - Primero codificamos la parte entera por divisiones sucesivas:
    - → $239_{)10} = 11101111_{)2}$



  - Después convertimos la parte decimal mediante multiplicaciones sucesivas:
    - → $0{,}403_{)10} = 0{,}0110011_{)2}$

0,403 * 2 = 0,806 (obtenemos un 0)
0,806 * 2 = 1,612 (obtenemos un 1)
0,612 * 2 = 1,224 (obtenemos un 1)
0,224 * 2 = 0,448 (obtenemos un 0)
0,448 * 2 = 0,896 (obtenemos un 0)
0,896 * 2 = 1,792 (obtenemos un 1)
0,792 * 2 = 1,584 (obtenemos un 1)
…

  - El número en binario será:
    - → $239{,}403_{)10} = 11101111{,}0110011\ldots_{)2}$

- Reflexiones: ¿Cuántos decimales he de obtener?
  ¿Cómo represento la coma en el ordenador?

- **Representación en coma flotante:**
  - Consiste en colocar la coma en una posición determinada, de manera que la representación binaria no se tendrá que preocupar por ella.
    - → $239{,}403_{)10} = 0{,}239403 * 10^{3}$
  - El hecho de mover la coma supondrá la aparición de una base elevada a un exponente, que también se ha de representar.
  - Cualquier número real en cualquier base se puede poner en coma flotante moviendo la coma a la izquierda de la primera cifra significativa del número y multiplicándolo por la base elevada al exponente adecuado.

$0{,}00000345_{)10} = 0{,}345 * 10^{-5}$

$11101111{,}011001_{)2} = 0{,}11101111011001 * 2^{8}$

- **Representación en coma flotante (continuación I):**
  - Una vez tenemos el número binario en coma flotante, la representación se hará separando por una parte la representación del número que aparece después de la coma (o mantisa) y, por otra, el exponente al que se eleva la base.
  - La mantisa se escribe representando los valores que aparecen a la derecha de la coma (ya que el 0 y la coma se mantienen fijos).
  - Como en la representación de enteros, tendremos que conocer el número de bits que hemos de utilizar en la representación.
  - La representación binaria de un número real en coma flotante siempre (o casi siempre) estará representada por:

    signo / mantisa / exponente

- **Representación en coma flotante (continuación II):**
  - La representación del signo y la mantisa se parece a la de los números enteros.
  - Por tanto, la forma habitual de representarlos (aunque se podría hacer en C1 o C2) es mediante signo y magnitud.
  - El exponente es un también es un número entero, de manera que también serviría cualquier método de representación de enteros para representarlo.
  - No obstante, es habitual que en la representación de números reales en coma flotante el exponente se represente con "sesgo" (es decir, con un desplazamiento o exceso), de manera que a los números negativos les hacemos corresponder números positivos. La transformación a aplicar es:

$$X' = X + 2^{n-1} \qquad X \in [-(2^{n-1} - 1), 2^{n-1}]$$

- **Representación en coma flotante (continuación III):**
  - Ejemplo: Representación en coma flotante, con 8 bits para la mantisa (en signo-magnitud) y 5 bits para el exponente ("sesgado"), del número:

    $239{,}403_{)10} = 11101111{,}0110011_{)2} = 0{,}11101111011011 * 2^8$

    Magnitud = 11101111010011 15 bits !!

    Mantisa = Signo positivo (0) + 7bits (1110111) = 01110111

  - Como el signo con la magnitud ocupan más de 8 bits, tendremos que **truncar** (perdida de información menos significativa.) Realmente sólo podremos representar el 0,1110111, en vez de 0,11101111010011.

    Exponente: Valor: $8_{)10} = 1000_{)2}$
    Sesgo (5 bits para el exponente → n = 5): $2^{n-1}{}_{)10} = 16 = 10000_{)2}$
    Exponente con sesgo será (exp = exp + 2^{n-1}):
    exp = 1000 + 10000 = 11000

  - **Finalmente: $239{,}403_{)10}$ = 01110111/11000**

Operaciones en coma flotante: Multiplicación

- Supongamos dos números A y B representados en coma flotante. En esa representación tendremos que:

  $A = mA * 2^{eA}$ $B = mB * 2^{eB}$

  Para multiplicar A y B, lo que haremos será:

  $A * B = (mA * 2^{eA}) * (mB * 2^{eB}) = (mA * mB) * 2^{eA+eB}$

- **Ejemplo:** Multiplicar los números 1,19 y 0,36 en coma flotante (8 bits para la mantisa en signo/magnitud y 5 para el exponente sesgado).

  $A = 1,19)_{10} = 1,0011)_2 = 0,1001100... * 2^1$

  $B = 0,36)_{10} = 0,01011100001)_2 = 0,1011100... * 2^{-1}$

  Exponente1: Valor: $1)_{10} = 00001)_2$
  Exponente2: Valor: $-1)_{10} = -00001)_2$
  Sesgo (5 bits para el exponente $\rightarrow$ n = 5): $2^{n-1})_{10} = 2^4)_{10} = 10000)_2$
  Exponente con sesgo será (exp = 2^{n-1}+ exp):
  exp1 = 10000 + 00001 = 10001
  exp2 = 10000 - 00001 = 01111

Multiplicación (continuación)

$A = 1{,}19)_{10} = 0{,}1001100 * 2^{1} = \underline{0}\ 1\ 0\ 0\ 1\ 1\ 0\ 0 / 1\ 0\ 0\ 0\ 1$

$B = 0{,}36)_{10} = 0{,}1011100 * 2^{-1} = \underline{0}\ 1\ 0\ 1\ 1\ 1\ 0\ 0 / 0\ 1\ 1\ 1\ 1$

- Multiplicamos las mantisas y ajustamos a la representación:

```
        0, 1 0 0 1 1 0
  x     0, 1 0 1 1 1 0
  ---------------------
          0 0 0 0 0 0
        1 0 0 1 1 0
      1 0 0 1 1 0
    1 0 0 1 1 0
  0 0 0 0 0 0
1 0 0 1 1 0
-----------------------
0,0 1 1 0 1 1 0 1 0 1 0
```

$A*B = 0{,}0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1 * 2^{1} * 2^{-1}$
$= (0{,}1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1 * 2^{-1}) * 2^{1} * 2^{-1}$

- Sumamos los exponentes

$A * B = 0{,}1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1 * 2^{-1} * 2^{1} * 2^{-1} = 0{,}1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1 * 2^{-1+1-1}$
$= 0{,}1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1 * 2^{-1}$
$= \underline{0}\ 1\ 1\ 0\ 1\ 1\ 0\ 1 / 0\ 1\ 1\ 1\ 1$ existe truncamiento!!!

Operaciones en coma flotante: Suma

- Sean los dos números A y B representados en coma flotante:
  $A = mA * 2^{eA}$ $B = mB * 2^{eB}$

Para poder sumar A y B tendremos que buscar qué número que multiplica se puede "sacar factor común" y sumar lo que queda:

Si **eA > eB** entonces significa que **eB = eA – eaux**

$$A + B = (mA * 2^{eA}) + (mB * 2^{eB}) = (mA * 2^{eA}) + (mB * 2^{eA-eaux}) =$$
$$= (mA + mB * 2^{-eaux}) * 2^{eA}$$

- **Ejemplo:** Sumar los números 1,19 y 0,36 en coma flotante (8 bits para la mantisa en signo/magnitud y 5 para el exponente sesgado).

$A = 1,19)_{10} = 1,0011)_2 = 0,1001100... * 2^1$

$B = 0,36)_{10} = 0,01011100001)_2 = 0,1011100... * 2^{-1}$

- **Suma (continuación)**

$A = 1,19)_{10} = 0,1001100 * 2^1 = \underline{0}\ 1\ 0\ 0\ 1\ 1\ 0\ 0 / 1\ 0\ 0\ 0\ 1$

$B = 0,36)_{10} = 0,1011100 * 2^{-1} = \underline{0}\ 1\ 0\ 1\ 1\ 1\ 0\ 0 / 0\ 1\ 1\ 1\ 1$

- Como expA > expB → expB = expA – aux = 1 – (2) = -1
- Reescribimos B:

$B = 0,1011100 * 2^{-1} = 0,1011100 * 2^1 * (2^{-1} * 2^{-1}) = 0,001011100 * 2^1 = 0,0010111 * 2^1$

- Ahora podemos sumar las mantisas y ajustar la representación:

$$\begin{array}{rr} & 0,\ 1\ 0\ 0\ 1\ 1\ 0\ 0 \\ + & 0,\ 0\ 0\ 1\ 0\ 1\ 1\ 1 \\ \hline & 0,\ 1\ 1\ 0\ 0\ 0\ 1\ 1 \end{array}$$

$A+B = 0,\ 1\ 1\ 0\ 0\ 0\ 1\ 1 * 2^1$

En este caso, la coma está bien colocada y no es necesario ajustar

- Representamos el exponente que era común y queda:

$A + B = \underline{0}\ 1\ 1\ 0\ 0\ 0\ 1\ 1 / 1\ 0\ 0\ 0\ 1$ También ha existido truncamiento!!!

Guión del tema

- Representación binaria de la información
  - Definición
  - Agrupación de bits
  - Unidades de medida binarias
- Información básica a representar
  - Información booleana
  - Conversión decimal – binario
  - Operaciones en binario
  - Enteros con signo
  - Caracteres
  - Reales
- Representación intermedia: octal y hexadecimal
  - Definición
  - Conversión decimal – hexadecimal/octal - binario
- Ejercicios

Representación intermedia (1/3)

- Como en base binaria los únicos dígitos que pueden utilizarse son el 0 y el 1, normalmente es engorroso y largo escribir un número en base binaria, aunque sea ésta la única base que realmente utiliza el ordenador.

- Para escribir números fácilmente convertibles a binario, pero con menor número de cifras se utilizan dos tipos de códigos intermedios: la base octal y la base hexadecimal.

- En la representación octal la base es ocho, y el alfabeto está formado por los dígitos entre 0 y 7.

B = 8
A = {0, 1, 2, 3, 4, 5, 6, 7}

Representación intermedia (2/3)

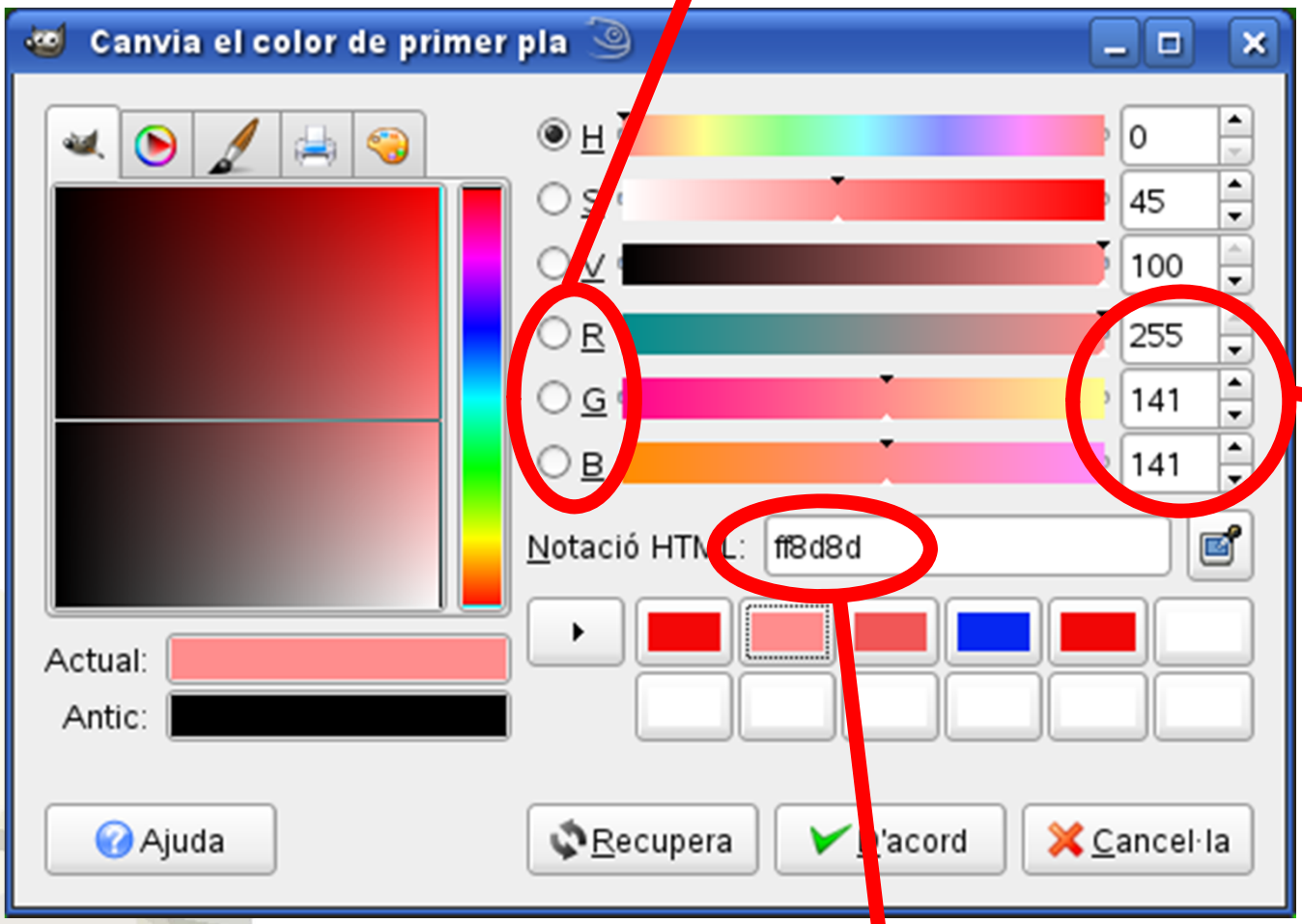
- En la representación hexadecimal la base es dieciséis, y el alfabeto, que constará de dieciséis caracteres, está formado por los dígitos desde el 0 hasta el 9 (diez caracteres) más las letras (generalmente mayúsculas) desde la A hasta la F (6 caracteres)

  B = 16
  A = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

- Por ejemplo, ésta representación es la que se usa a la hora de definir los colores en cualquier herramienta de dibujo.

Representación intermedia(3/3)

Representación RGB: Red, Green, Blue

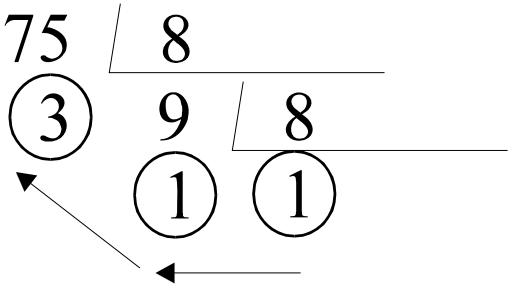


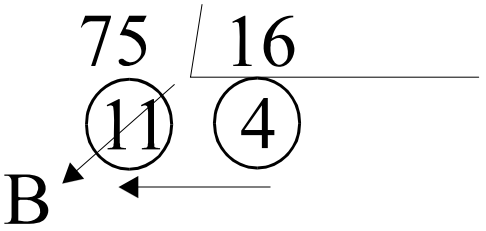
Representación decimal

Representación hexadecimal

Conversión decimal – octal/hexadecimal

- La manera de pasar de base decimal a octal o hexadecimal es similar al paso a base binaria, pero dividiendo en cada caso sucesivamente por ocho o dieciséis.





- Por ejemplo: $75_{)10} = 113_{)8} = 4B_{)16}$
- Para cambiar de octal/hexadecimal a decimal, al igual que en binario, aplicaremos la fórmula de descomposición:
  - Octal= suma de potencias de 8
  - Hexadecimal = suma de potencias de 16
- Por ejemplo, para pasar $4B_{)16}$ y $113_{)8}$ a base decimal:

$$113_{)8} = 3*8^0+1*8^1+1*8^2 = 3*1+1*8+1*64 = 3+8+64 = 75$$

$$4B_{)16} = B*16^0+4*16^1 = 11*1+4*16 = 11+64 = 75$$

Conversión binaria – octal

- La base octal está basada en el 8, mientras que la binaria esta basada en el 2.
- Como $8 = 2^3$ podemos decir que tres cifras binarias hacen una cifra octal.
- Por tanto, el paso de binario a octal se reduce a agrupar de tres en tres, de derecha a izquierda, las cifras binarias y evaluar su valor decimal. Recuerda rellenar con ceros a la izquierda hasta completar agrupaciones de 3.

$\mathbf{1001011}_{)2} = 001\ 001\ \ 011_{)2}$

$001_{)2} = 1_{)8}$

$001_{)2} = 1_{)8}$

$011_{)2} = 3_{)8}$

$= \mathbf{113}_{)8}$

Conversión binaria – hexadecimal

- La representación hexadecimal está basada en el 16, mientras que la binaria está basada en el 2.
- Como $16 = 2^4$ podemos asumir que cuatro cifras binarias hacen una cifra hexadecimal.
- Así, el paso de binario a hexadecimal se reduce a agrupar de cuatro en cuatro, de derecha a izquierda, les cifras binarias y evaluar el valor decimal; cabe recordar que valores superiores a 9 se representan por medio de letras (A=10, B=11, C=12, D=13, E=14 y F=15)

$\mathbf{1001011}_{)2} = 0100\ 1011_{)2}$

$0100_{)2} = 4_{)16}$

$1011_{)2} = 11 = B_{)16}$

$= \mathbf{4B}_{)16}$

Conversión octal/ hexadecimal - binario

- Una vez visto el paso de binario a octal y hexadecimal, la transformación inversa es obvia.

- En el caso octal sólo hay que pasar cada uno de los dígitos octales a tres cifras binarias y en el caso hexadecimal a cuatro cifras binarias.

$113_{)8} = 001\ 001\ 011_{)2}$

$1_{)8} = 001_{)2}$

$3_{)8} = 011_{)2}$

$4B_{)16} = 0100\ 1011_{)2}$

$4_{)16} = 0100_{)2}$

$B_{)16} = 11 = 1011_{)2}$

Bin	Hex	Dec
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

octal

Ejercicios

- Pasar a binario los números y las letras siguientes:
  - $-87_{)10}$ ---> $?_{)2SM}$
  - $01110000_{)2}$ --> $?_{)16}$ --> $?_{)ASCII-8}$
  - $t_{)ASCII-8}$ --> $?_{)16}$--> $?_{)2}$
  - $166{,}386_{)10}$ --> $?_{)2}$

- Calcular la representación hexadecimal o decimal de los colores:
  - Blanco : R(?), G(?), B(?)
  - Negro : R(?), G(?), B(?)
  - Gris : $969696_{)16}$ → R(?), G(?), B(?)
  - Turq. : $22e0bf_{)16}$ → R(?), G(?), B(?)