1
CODIFICACIÓN DE LA INFORMACIÓN

Sistemas operativos y lenguajes de programación

CF GS Sistemas de telecomunicación e informáticos. 1er curso
Curso 2012-13

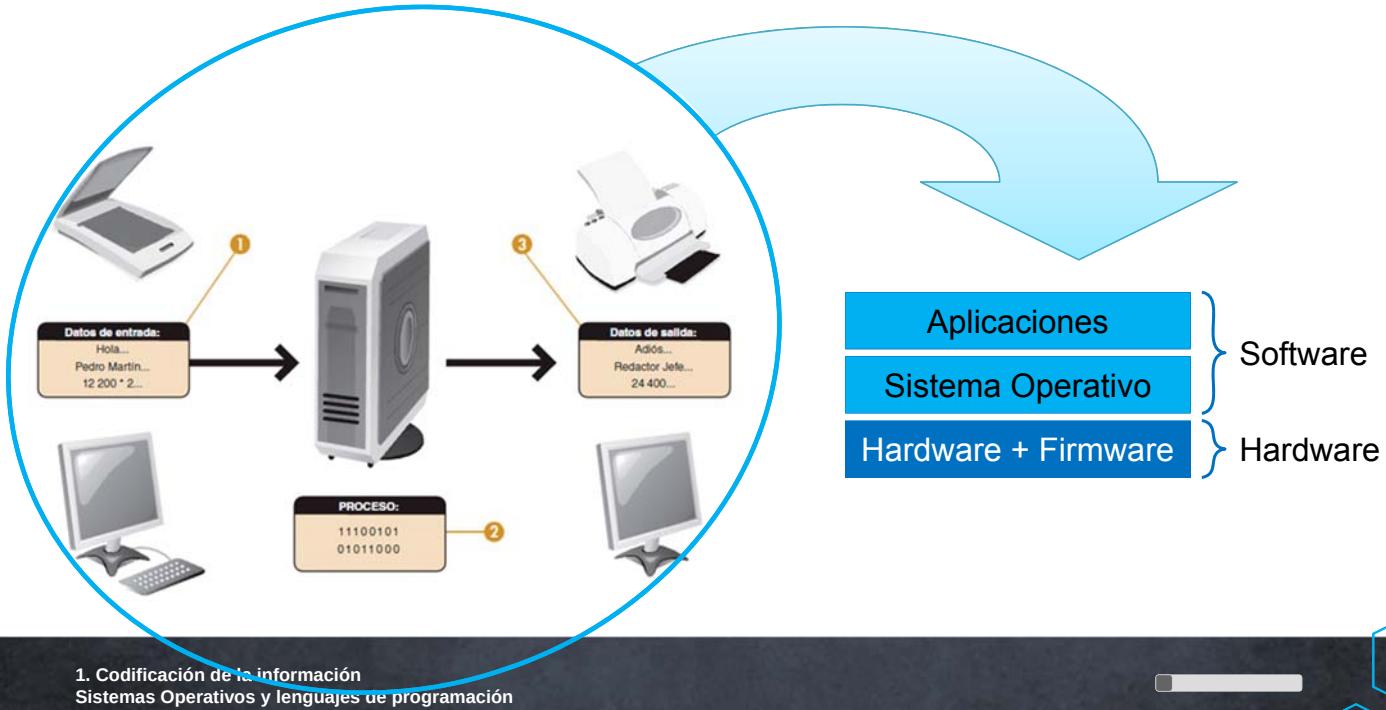
por: JCOL

SUMARIO

11010011

1. ESTRUCTURA LÓGICA DE UN ORDENADOR

Ciclo básico del procesamiento de datos:



2. TIPOS DE DATOS

- La utilidad del sistema informático es procesar información. La información a procesar (datos de entrada) puede ser de diferentes tipos: textos, gráficos, imágenes, audio, video.
- A su vez el SW del computador para procesar los datos de entrada necesitará tratar la información (datos intermedios). Como resultado del proceso obtendremos una nueva información (datos de salida).
- Toda la información que estamos tratando la tendremos que representar en un formato que pueda entender el computador (proceso de codificación). Hay que llegar a un compromiso entre coste de traducción, coste de almacenamiento y coste de tratamiento. Para ganar en fiabilidad se emplean códigos redundantes. Para aumentar la seguridad se recurre a la encriptación y para ahorrar espacio de almacenamiento se usan métodos de compactación.
- Los datos se implementan sobre códigos:
  - Numéricos: dígitos del 0 al 9
  - Alfabéticos: letras mayúsculas y minúsculas, de la A a la Z.
  - Alfanuméricos: numéricos + alfabéticos + caracteres especiales (*, /, -, %, etc.)
  - Otros específicos para imágenes, audio,… (pero siempre en formato binario).

3. CÓMO MEDIR LA INFORMACIÓN

La cantidad mínima de información es un bit [b] (*BInary digiT*), que representa 2 posibles estados:

- 0 ó 1
- Sí o No
- Verdadero o Falso

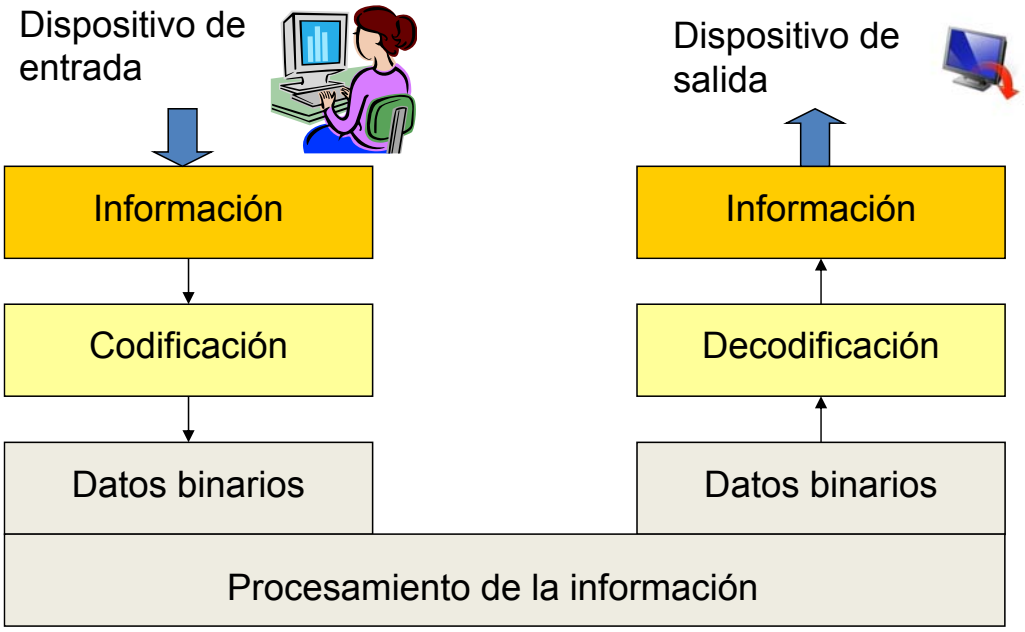
Un byte [B] es el nº de bits necesario para representar un carácter (8 bits).

1 Kilobyte (KB)	= 2^{10} Bytes	= 1.024 Bytes	≈ 10^{3} Bytes
1 Megabyte (MB)	= 2^{20} Bytes	= 1.048.576 Bytes	≈ 10^{6} Bytes
1 Gigabyte (GB)	= 2^{30} Bytes	= 1.073.741.824 Bytes	≈ 10^{9} Bytes
1 Terabyte (TB)	= 2^{40} Bytes	= 1.099.511.627.776 Bytes	≈ 10^{12} Bytes
1 Petabyte (PB)	= 2^{50} Bytes	= 1.125.899.906.842.624 Bytes	≈ 10^{15} Bytes
1 Exabyte (EB)	= 2^{60} Bytes	= 1.152.921.504.606.846.976 Bytes	≈ 10^{18} Bytes

Denominación y equivalencia entre los múltiplos del byte

4. SISTEMAS DE CODIFICACIÓN

El Sistemas Operativo y el hardware se encargan de codificar y decodificar la información.



Ejemplo: El SO muestra un carácter diferente en pantalla según el sistema de codificación seleccionado.

5. SISTEMAS DE NUMERACIÓN

Un sistema de numeración es el conjunto de símbolos y reglas que se utilizan para representar cantidades o datos numéricos.

Los sistemas de numeración que utilizamos son sistemas posicionales → el valor relativo que cada símbolo representa queda determinado por su valor absoluto (X) y por su posición (i).

El número de símbolos distintos de un sistema lo determina la base (B)

- Sistema decimal (base 10) ⇒ símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9.
- Sistema octal (base 8) ⇒ símbolos: 0, 1, 2, 3, 4, 5, 6 y 7.
- Sistema binario (base 2) ⇒ símbolos: 0 y 1

Así en un sistema posicional, el valor de una cantidad viene determinada por la fórmula:

$$N = \Sigma\, X_i\, B^i$$

Ejemplo: $458 \text{ (base 10)} = 458_{(10} = 4 \cdot 10^2 + 5 \cdot 10^1 + 8 \cdot 10^0 = 4 \cdot 100 + 5 \cdot 10 + 8 \cdot 1$

6. CODIFICACIÓN NUMÉRICA

En los sistemas informáticos se suelen emplear tres sistemas de codificación para los números:

- Binario (base 2), símbolos: 0 y 1 ← cada símbolo equivale a 1 bit
- Octal (base 8), símbolos del 0 al 7.
- Hexadecimal (base 16), símbolos del 0 al 9 y de la A a la F.

Hay correspondencia de los sistemas octal y hexadecimal con el binario.

- Cada símbolo octal → 3 bits.
- Cada símbolo hexadecimal → 4 bits.

El sistema trabaja internamente en binario pero a la hora de mostrar información, ésta se presenta en octal y hexadecimal. Como ejemplo el SO suele mostrar las direcciones de memoria en hexadecimal.

HEXADECIMAL BASE 16	DECIMAL BASE 10	OCTAL BASE 8	BINARIO BASE 2
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111

7. CAMBIOS DE BASE DE NUMERACIÓN (I)

Cambios de base:

- De base 10 a base 2 → Se divide sucesivas veces el número en base 10 entre 2 hasta que el cociente es 0. Los restos resultantes de la división en orden inverso forman el número en base 2.
  - Ejemplo: Pasar $90_{(10}$ a base 2:

```
90 | 2
 0 | 45 | 2
     1 | 22 | 2
         0 | 11 | 2
             1 |  5 | 2
                 1 |  2 | 2
                     0 |  1 | 2
                         1 |  0
```

  Resultado: $90_{(10} = 1011010_{(2}$

- Base 10 a base 8 → Se divide sucesivas veces el número en base 10 entre 8 hasta que el cociente es 0. Los restos resultantes de la división en orden inverso forman el número en base 8.
  - Ejemplo: Pasar $90_{(10}$ a base 8:

```
90 | 8
 2 | 11 | 8
     3 |  1 | 8
         1 |  0
```

  Resultado $90_{(10} = 132_{(8}$

7. CAMBIOS DE BASE DE NUMERACIÓN (II)

- De base 10 a base 16 → Se divide sucesivas veces el número en base 10 entre 16 hasta que el cociente es 0. Los restos resultantes de la división en orden inverso forman el número en base 16.
  - Ejemplo: Pasar $90_{(10}$ a base 16:

```
90 | 16
10 |  5 | 16
     5 |  0
```

```
10 = A
```

  Resultado: $90_{(10} = 5A_{(16}$

- Base 2 a base 10 → Se aplica la fórmula $N = \sum X_i B^i$
  - Ejemplo: $1001_{(2}$ a base 10.

$$1001_{(2} = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 0 + 0 + 1 = 9$$

  Resultado $1001_{(2} = 9_{(10}$

7. CAMBIOS DE BASE DE NUMERACIÓN (III)

Cuando las bases son equivalentes, cosa que se puede saber si una base se puede representar como una potencia de la otra, los cambios se simplifican.

Esta transformación se realiza de forma directa formando grupos de bits según la relación de potencia entre las bases:

- $8 = 2^3 \rightarrow$ grupos de 3 bits
- $16 = 2^4 \rightarrow$ grupos de 4 bits
  - Ejemplo: Pasar $132_{(8}$ a base 16.

Primero hacemos la conversión de base 8 a base 2:

$1_{(8} = 001_{(2}$
$3_{(8} = 011_{(2}$ $\Rightarrow 132_{(8} = 001011010_{(2}$
$2_{(8} = 010_{(2}$

Añado tres ceros a la izquierda para poder formar grupos de 4 dígitos: 0000 0101 1010

0 5 C

Resultado final $132_{(8} = 5C_{(16}$

7. CAMBIOS DE BASE DE NUMERACIÓN (Y IV)

Actividad 1.1

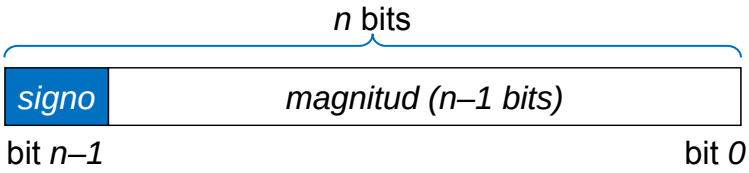
Completa el siguiente cuadro:

DECIMAL	BINARIO	OCTAL	HEXADECIMAL	BASE 5
1021				
	101101101			
		5342		
			F8A	

8. REPRESENTACIÓN DE NÚMEROS ENTEROS CON SIGNO (I)

Siempre se busca un compromiso entre el rango de valores, la precisión, la facilidad de operar y el nº de bits que necesita el sistema de representación.

Hay varios métodos, pero el más sencillo es añadir un bit extra a la izquierda del número en valor absoluto.



- Si el bit (*n–1*) que representa el signo tiene el valor:
  - 1 ⇒ el número es negativo.
  - 0 ⇒ el número es positivo.

Este sistema se llama binario puro con coma o punto fijo. En la práctica se emplean 32 bits (1 bit de signo y 31 para el valor del número).

- Ejemplo: el número –10 = 10000000000000000000000000001010

8. REPRESENTACIÓN DE NÚMEROS ENTEROS CON SIGNO (II)

Veamos otros tipos de codificación numérica, pero para ello, antes necesitamos saber cómo se representan los números decimales en BCD (*Bynary Coded Decimal*), en el que cada dígito es una combinación de 4 bits (cuarteto).

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

Otros tipos:

- Decimal desempaquetado: representa cada número decimal de forma que:
  - cada una de sus cifras ocupa un byte (octeto = 2 cuartetos).
    - 1er cuarteto (cuarteto de la izquierda): *bits de zona*: 1111
    - 2º cuarteto (cuarteto de la derecha): número en BCD.
  - El signo se representa en el cuarteto de bits de la izquierda (correspondiente a los bits de zona) de la última cifra:
    - Signo positivo: 1100
    - Signo negativo: 1101
  - Ejemplo: Representar +2371 y –2371

	2		3		7		1	
+2371:	1111	0010	1111	0011	1111	0111	1100	0001
–2371:	1111	0010	1111	0011	1111	0111	1101	0001

8. REPRESENTACIÓN DE NÚMEROS ENTEROS CON SIGNO (Y III)

- Decimal empaquetado: representa cada cifra con un conjunto de 4 bits (un cuarteto) y se añaden 4 bits a la derecha para representar el signo con el mismo criterio que para el desempaquetado:
  - Ejemplo: Representar +2371 y –2371:

	2	3	7	1	
+2371:	0010	0011	0111	0001	1100
–2371:	0010	0011	0111	0001	1101

9. CODIFICACIÓN DE NÚMEROS REALES: PARTE ENTERA Y DECIMAL

Si tenemos que representar números enteros muy grandes o que tengan parte decimal se recurre a los sistemas conocidos como de coma flotante:

- Se dividen los n bits disponibles para representar un dato en dos partes llamadas mantisa (*M*) y exponente (*E*).
- La mantisa contiene los datos significativos del dato y el exponente el factor de escala.
- La cantidad se obtiene como: $M \cdot r^E$, donde *r* es la base (normalmente 2).

El estándar IEEE 754 define la implementación de representaciones en coma flotante de 2 formas:

- Simple precisión (32 bits)
- Doble precisión (64 bits)

signo	*exponente (8 bits)*	*mantisa (23 bits)*
signo	*exponente (11 bits)*	*mantisa (52 bits)*

10. CODIFICACIÓN ALFANUMÉRICA (I)

Representan caracteres alfanuméricos: cifras, letras, caracteres especiales y caracteres de control.

Los más usados ahora son:

- ASCII (*American Standard Code for Information Interchange*), con 7 u 8 bits para representar hasta 256 símbolos.
- EBCDIC (*Extended BCD Interchange Code*), de 8 dígitos en dos bloques de 4.
- UNICODE. Es el código internacional utilizado actualmente en la mayoría de los SO. Presenta la ventaja de que proporciona un número único para cada carácter, con independencia del software, hardware o el idioma. ASCII sin embargo necesita una tabla para cada país ya que los símbolos de todos los países no se pueden codificar con sólo 8 bits.

Estos sistemas de codificación se crean con reglas que facilitan su tratamiento. Por ejemplo en ASCII (ver tabla de la transparencia siguiente):

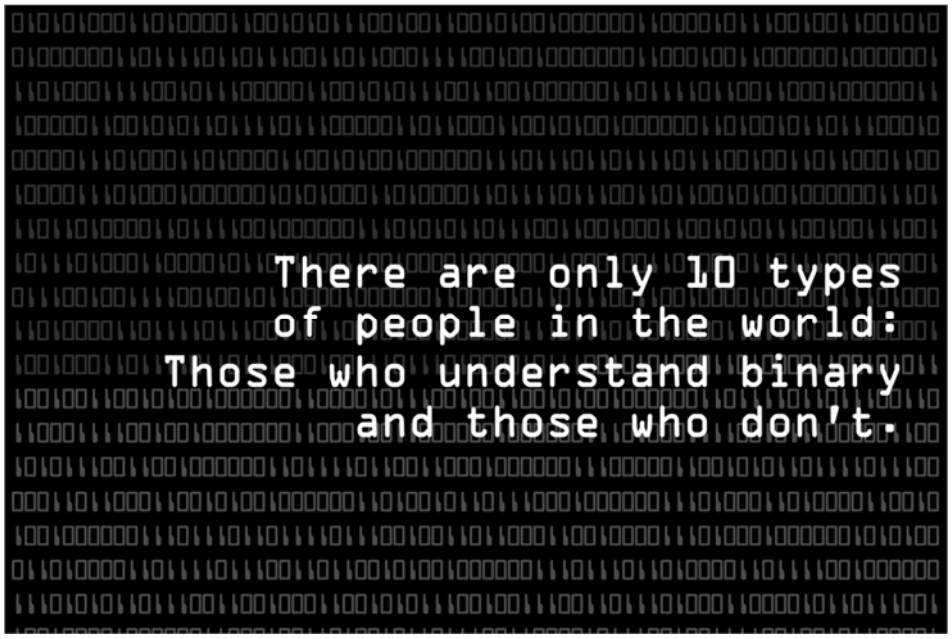
1. Los caracteres numéricos están seguidos. Para conocer su valor en ASCII se resta 48.
2. Las letras mayúsculas y minúsculas sólo difieren en un bit (así es fácil la conversión.

10. CODIFICACIÓN ALFANUMÉRICA (Y II)

Binario	Dec	Hex	Represent.
0010 0000	32	20	espacio ( )
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

Binario	Dec	Hex	Represent.
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D	]
0101 1110	94	5E	^
0101 1111	95	5F	_

Binario	Dec	Hex	Represent.
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q
0111 0010	114	72	r
0111 0011	115	73	s
0111 0100	116	74	t
0111 0101	117	75	u
0111 0110	118	76	v
0111 0111	119	77	w
0111 1000	120	78	x
0111 1001	121	79	y
0111 1010	122	7A	z
0111 1011	123	7B	{
0111 1100	124	7C	|
0111 1101	125	7D	}
0111 1110	126	7E	~



FIN DEL TEMA 1

CODIFICACIÓN DE LA INFORMACIÓN