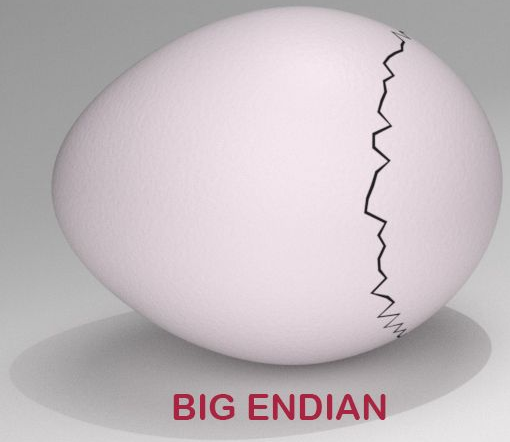
Codificació

2a part

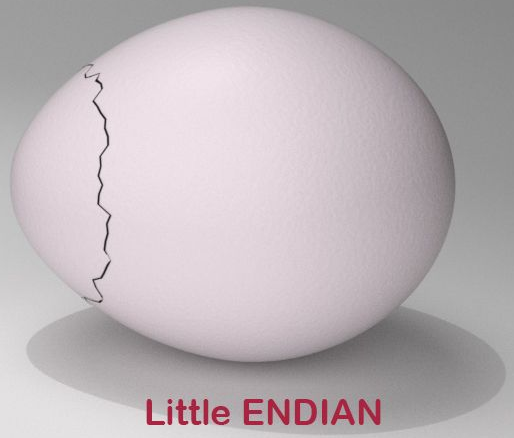
Endianness

The way traditionally lilliputians broke their boiled eggs on the larger end



BIG ENDIAN

The way the king then ordered lilliputians to break their boiled eggs on the smaller end



Little ENDIAN

Endianness

L’ordre dels bytes pot ser:

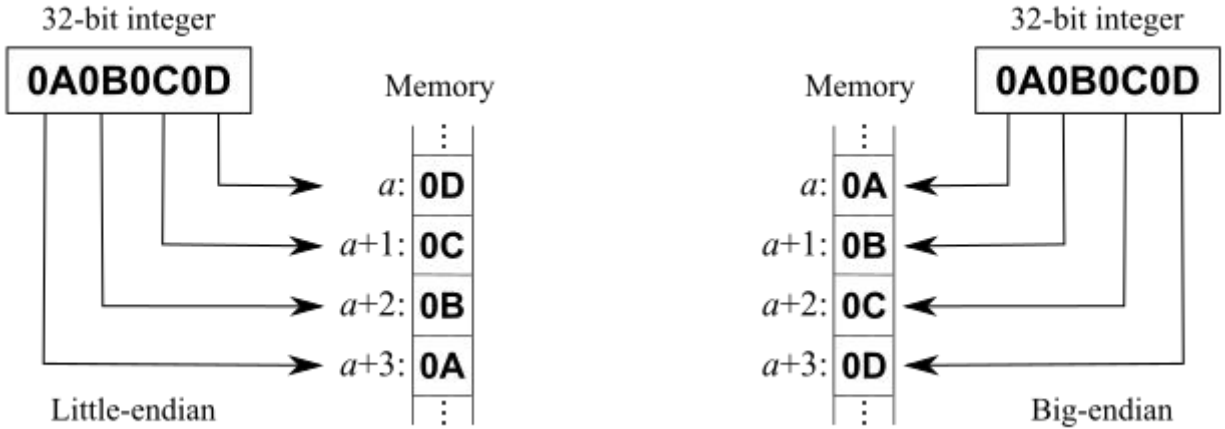
- Little-Endian (Little-End-In), Intel, AMD.
- Big-Endian (Big-End-In), Network order, used in Internet comunications, IBM, SPARC, Motorola.

Alguns processadors soporten Bi-endianness, com els nous ARM.

Altres processadors incorporen funcions de conversió (Intel: bswap; ARM: rev).

Ordre dels bytes: Big-Endian i Little-Endian

L'ordre dels bytes afecta tant a les dades com als mateixos programes i les seves instruccions.



Endianness (ordre dels bytes)

Alguns processadors soporten bi-endianness (pe. els ARM més nous), però es donen diferents situacions de bi-endianness:

- Només per dades, o per les dades i els programes.
- Configurable des de programari, o configuració del processador en la placa mare.

La característica d'ordre dels bytes d'un processador representa un **problema** quan s'han d'**intercanviar arxius**; el problema no es dona en ASCII, perquè és una codificació només en 1 byte.

Solucions endianness per l'intercanvi d'arxius

1. Fixar l'ordre dels bytes com a part de l'especificació del format de l'arxiu.
    - És el cas d'*arxius XSL*, que són Little-Endian i requereixen conversió en els Big-Endian.
    - El *sistema d'arxius* **FAT** és Little-Endian, independentment del processador que el faci servir.
2. Indicar l'ordre dels bytes en l'arxiu.
    - Les imatges ***TIFF*** comencen amb **II** per a Little-Endian i per **MM** els Big-Endian; *II* es refereix a *Intel*, i *MM* a *Motorola*; el codi es palíndrom perquè sigui llegit igual en tots dos sistemes.
    - Els arxius en ***UTF*** incorporen un prefix anomenat **BOM** (*Byte Order Mark*), que indica l'ordre dels bytes.

UTF BOM (Byte Order Mark)

El primer caràcter d'un arxiu UTF és el BOM

UTF-16:

- Big-Endian: `FE FF`. (visualitzat com a þÿ)
- Little-Endian: `FF FE`. (visualitzat com a ÿþ)

UTF-32:

- Big-Endian: `00 00 FE FF`.
- Little-Endian: `FF FE 00 00`.

Cal anotar que ni FFFE ni FEFF són caràcters vàlids en UTF-16, però tampoc FF ni FE són caràcters vàlids en UTF-8, així que el BOM també permet detectar que l'arxiu és de codificació UTF-16.

El caràcter 00 és el caràcter *NULL*, que tampoc és un caràcter vàlid, així que també es pot detectar que l'arxiu està codificat en UTF-32.

En UTF-32 es pot ometre el BOM quan és Big-Endian, però també si són arxius que gestiona una aplicació, pe. arxius de configuració.

UTF-8 BOM

En UTF8 el BOM són 3 bytes amb el valor `EF BB BF` (es visualitza com a **ï»¿**).

No es requereix fer-ho servir, però actua com a *magic-number*, per reconèixer la codificació UTF-8, una pràctica habitual en els sistemes basats en Unix.

La diferència amb un arxiu ASCII consistirà en els caràcters no codificables en els 128 primers caràcters ASCII (accents, dièresi, Ç, ela geminada, caràcters d'altres idiomes...)

- Si un arxiu UTF8 s'interpreta com a ASCII aquests caràcters seran il·legibles.
- Si un arxiu de UTF8 té format preparat per processar, si és llegit com a ASCII probablement provocarà errors quan sigui processat.

Base64, Radix64 i UTF7: Codificació de binari a text

De vegades és necessari incorporar un arxiu binari en un arxiu de text, per exemple per incorporar una imatge en comptes de tenir-la en un altre arxiu.

La codificació Base64 permet **codificar arxius binaris com a text**.

El text resultant es divideix en línies de longitud fixe, separades amb els caràcters de final de línea (CR+LF).

- Radix64 incorpora una verificació final (CRC, Control de redundància cíclica)
- UTF7 mai reconegut, en desús. No dividia en línies el text resultant.

Codificació Base64

Ens limitem a 64 símbols, de forma que tots són caràcters imprimibles: 26 lletres majúscules i 26 de minúscules, 10 dígits i els símbols ‘+’ i ‘/’.

Cada 3 bytes (24 bits) generarem 4 caràcters, fent servir 6 bits per a cada caràcter, 64 valors diferents.

A més es fa servir el símbol “=” com a “*padding*”, per fer el nombre final de caràcters resultants divisible entre 3.

Base64 Encoding Table

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Codificació

Text	**M**	**a**	**n**
ASCII	77	97	110
Bits	0100·1101	0110·0001	0110·1110

Índex	0100·11	01·0110	000101	101110
Índex	19	22	5	46
Base64	**T**	**W**	**F**	**u**

Es fan servir grups de 6 bits, obtenint un valor que fa d'index en la taula de base64, resultant un caràcter.

Per tant, cada 3 bytes es codifiquen en 4 bytes, que resulten en un caràcter imprimible cadascú.
L'exemple s'ha fet amb la paraula "*Man*", però podria ser qualsevol valor binari.

Text d’exemple:

Text codificat:

```
TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSB0aGlz
IHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBhIGx1c3Qgb2Yg
dGhlIG1pbmQsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2Ugb2YgZGVsaWdodCBpbiB0aGUgY29udGlu
dWVkIGFuZCBpbmRlZmF0aWdhYmxlIGdlbmVyYXRpb24gb2Yga25vd2xlZGdlLCBleGNlZWRzIHRo
ZSBzaG9ydCB2ZWhlbWVuY2Ugb2YgYW55IGNhcm5hbCBwbGVhc3VyZS4 =
```

Base64 encode/decode online tool

Utilitat:

- Actualment, per incorporar imatges en **HTML**, **CSS** i **SVG**
- El hash dels passwords dels sistemes tipus Unix es guarda en un arxiu de text (/etc/passwd i /etc/shadow) en Radix64.