Architettura degli Elaboratori 2025-2026

Rappresentazione numeri reali

Rappresentazione altre informazioni

Prof. Elisabetta Fersini
elisabetta.fersini@unimib.it

Rappresentazione di numeri reali

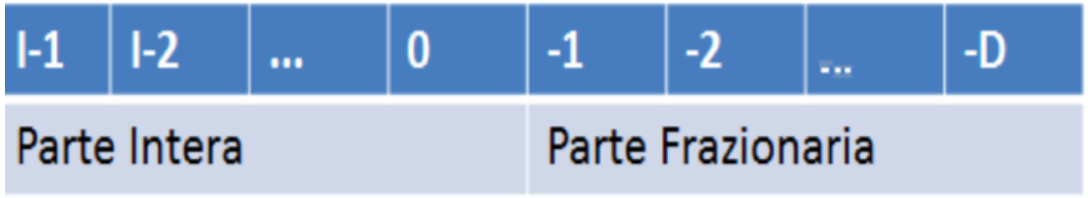
- I numeri reali possono essere rappresentati come segue:
  - Virgola fissa (*fixed point*)
  - Virgola mobile (*floating point*)

Virgola fissa

- Un sistema di numerazione in **virgola fissa** è quello in cui:
  - Si riserva un **numero fisso di bit** per parte intera e parte frazionaria;
  - La **posizione** della virgola decimale è **implicita**
  - La posizione della virgola decimale **uguale** in tutti i numeri

Virgola fissa

- **UNSIGNED** fixed point: dati N bit a disposizione
  - I<N bit per rappresentare la parte intera del numero
  - D=N-I bit per rappresentare la parte decimale del numero



Virgola fissa

- **UNSIGNED** fixed point: dati N bit a disposizione
  - I<N bit per rappresentare la **parte intera** del numero
  - D=N-I bit per rappresentare la **parte decimale** del numero

Con questo metodo l'intervallo di numeri interi rappresentabili è $[0, 2^I-1]$

L'intervallo rappresentabile dalla parte decimale è $[0, 2^D-1]$.

Virgola fissa

- **SIGNED** fixed point: dati N bit a disposizione
  - 1 bit per il **segno** del numero da rappresentare
  - I<(N-1) bit per rappresentare la **parte intera** del numero
  - D=N-(I+1) bit per rappresentare la **parte decimale** del numero

+/-	I-1	I-2	...	0	-1	-2	...	-D
Segno	Parte Intera				Parte Frazionaria			

Virgola fissa

- **SIGNED** fixed point: dati N bit a disposizione
  - 1 bit per il **segno** del numero da rappresentare
  - I<(N-1) bit per rappresentare la **parte intera** del numero
  - D=N-(I+1) bit per rappresentare la **parte decimale** del numero

Con questo metodo l'intervallo di numeri interi rappresentabili è $[-2^{I-1}-1, 2^{I-1}-1]$

L'intervallo rappresentabile dalla parte decimale è $[0, 2^{D}-1]$.

Virgola fissa

- Supponiamo di volere rappresentare numeri reali con 8 bit. Possiamo decidere di dedicare:
  - 1 bit al segno, 3 bit alla parte intera e 4 bit alla parte decimale.
  - Oppure 1 bit al segno, 5 bit alla parte intera e 2 bit alla parte decimale.
- La scelta è dettata dalle necessità pratiche.
  - Se volessimo rappresentare numeri che hanno una parte intera grande (e quindi che hanno un valore I grande) ma una precisione piccola (dedicando meno bit alla parte decimale del numero) sceglieremmo la **seconda opzione**.
  - Se volessimo rappresentare numeri che hanno parte intera piccola ma una grande precisione decimale, sceglieremmo la **prima opzione**.

Virgola fissa

- Consideriamo il numero X.YYYY in base 2. La conversione di un numero frazionario (base 2) in decimale (base 10) in virgola fissa avviene come ricorrendo alla conversione vista per il sistema binario puro.

- Esempio: consideriamo il numero 101.01_2, e rappresentiamolo in base 10

$$101.01_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 4 + 0 + 1 + 0 + 0.25 = 5.25_{10}$$

Virgola fissa

- Consideriamo il numero X.YYYY in base 10. La **conversione** di un numero frazionario (base 10) in binario (base 2) **in virgola fissa** avviene come segue:

  1. Si considera la parte intera X e la si riporta in base 2
  2. La parte decimale YYYY viene scomposta per moltiplicazioni successive, prendendo la parte intera di ciascun risultato

Virgola fissa

Esempio di conversione in virgola fissa da base 10 a base 2

Convertiamo il numero 9,6234

```
parte intera 9 -> 1001
```

```
parte dopo la virgola
0,6234 * 2 = 1,2468
0,2468 * 2 = 0,4936
0,4936 * 2 = 0,9872
0,9872 * 2 = 1,9744
0,9744 * 2 = 1,9488
0,9488 * 2 = 1,8976
0,8976 * 2 = 1,7952
0,7952 * 2 = 1,5904
...
```

Non bastano 8 cifre!

Errore di approssimazione

Virgola fissa

- **Svantaggi** della rappresentazione in virgola fissa:
  - Rigidità della posizione assegnata alla virgola
    - Sono fissi i bit assegnati per codificare la parte intera e la parte frazionaria
  - Impatta sulla precisione nel codificare i numeri
    - Maggiore è il numero di bit per codificare la parte intera, più bassa sarà la precisione nel codificare i numeri piccoli
  - Spreco di bit per memorizzare zeri
    - Sia in numeri molto grandi che in numeri molto piccoli, in presenza di molti zeri, occupo bit per doverli rappresentare

Virgola mobile

La massa dell'elettrone è
0.000000000000000000000000000000091 Kg

La massa della terra è
5973600000000000000000000 kg

Come possiamo gestire numeri di questo **tipo** e stabilire **univocamente** la convenzione sulla **posizione della virgola**?

Di quale **capacità di rappresentazione** avremmo bisogno rappresentando queste quantità in binario?

Virgola mobile

La massa dell'elettrone è
0.00000000000000000000000000000091 Kg

La massa della terra è
5973600000000000000000000 kg

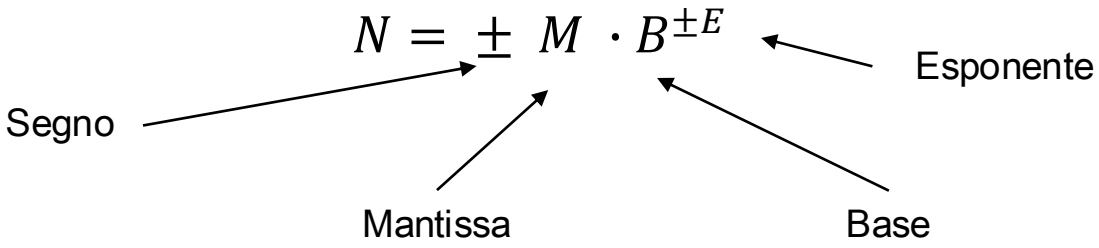
Potremmo semplicemente scrivere il numero indicandone le cifre essenziali

Ad esempio:

- $9.1 * 10^{-31}$
- $5.9736 * 10^{24}$.

Virgola mobile

- Questa notazione all'interno degli elaboratori si definisce virgola mobile (floating point)
- In forma generale, un numero N è esprimibile come



- Più bits per mantissa: maggior accuratezza

- Più bits per esponente: maggior intervallo

Virgola mobile

- Usa un bit per rappresentare il **segno *s***
- Usa altri bit per rappresentare la **mantissa *m***, detta anche *significante*
- Usa altri per codificare l'**esponente *e***

s	e	e	...	e	m	m	...	m
Segno	esponente				mantissa			

Virgola mobile

- Due forme di rappresentazione:

1. Non-normalizzata (arbitraria):

$363,4 \bullet 10^{34}$ $36,34 \bullet 10^{35}$

2. Normalizzata: 1 cifra (diversa da zero) per la mantissa

$\mathbf{3,634 \bullet 10^{36}}$ (base 10)

$\mathbf{1,xxx \bullet 2^{yy}}$ (base 2)

Virgola mobile

- Si consideri un numero X esprimibile in virgola mobile utilizzando un mantissa di 4 bit pari a M=1011, base B=2 ed esponente e=01010.

- A quanto corrisponde X in base decimale?

Virgola mobile

- Estende l'intervallo di numeri rappresentati a parità di cifre, rispetto alla notazione in *virgola fissa*
- I numeri reali con segno sono quindi rappresentati da :
  - Mantissa (M)
  - Esponente (E)
  - Segno (S)

Un numero X sarà scritto come

$$X = (-1)^S * M * B^E$$

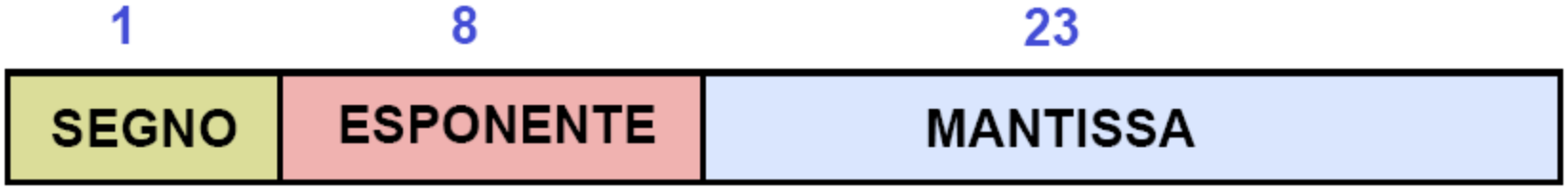
Virgola mobile

- La notazione scientifica per la base 2:

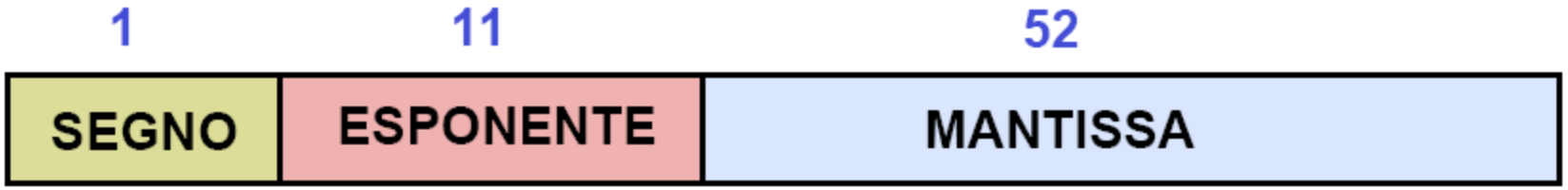
$$1,xx\ldots xx_2 \cdot 2^{yy\ldots yy_2}$$

dove le *x* rappresentano la parte frazionaria e le *y* l'esponente a cui elevare la base 2

- Semplice/singola precisione su 32 bit:



- Doppia precisione su 64 bit:

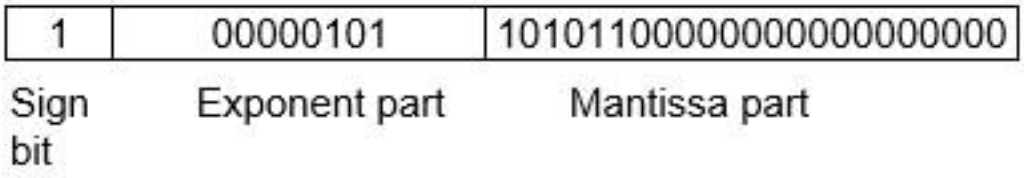


Virgola mobile

- Supponiamo che il numero utilizzi il formato a 32 bit:
  - il bit di segno a 1 bit
  - 8 bit per l'esponente con segno
  - 23 bit per la parte frazionaria.

Il bit iniziale 1 non viene memorizzato (poiché è sempre 1 per un numero normalizzato) e viene definito "bit nascosto".

−53.5 viene rappresentato «codificato» $-53.5=(-110101.1)_2=(-1.101011) \times 2^5$



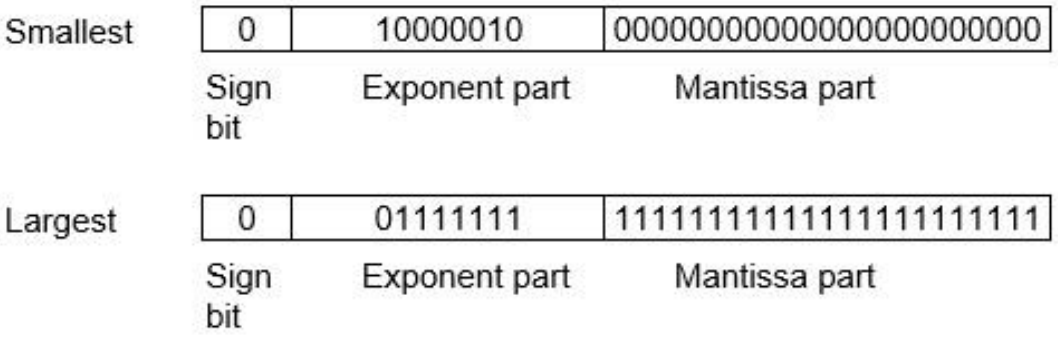
Virgola mobile

- Il numero **positivo** normalizzato **più piccolo** rappresentabile in 32 bit è

  $(1.00000000000000000000000)_2 \times 2^{-126} = 2^{-126} \approx 1.18 \times 10^{-38}$

- il numero **positivo** normalizzato **più grande** rappresentabile in 32 bit è

  $(1.11111111111111111111111)^2 \times 2^{127} = (2^{24}-1) \times 2^{104} \approx 3.40 \times 10^{38}$.



Standard IEEE 754

- Si è reso necessario definire uno standard per la rappresentazione dei numeri in virgola mobile per definire la semantica delle istruzioni in virgola mobile
- IEEE Computer Society (Institute of Electrical and Electronics Engineers) definisce lo "IEEE standard for binary floating arithmetic" noto anche come **IEEE 754** nel 1985.
  - Specifica il **formato**, le **operazioni**, le **conversioni** tra i diversi formati floating point e quelle tra i diversi sistemi di numerazione, il trattamento delle **eccezioni**
- Nel 1989 IEEE 754 diventa uno standard diventa uno **standard internazionale**

IEEE 754

- Formato non proprietario, ossia non dipendente dall'architettura del calcolatore
- **Esponente (8 bit)**
  - Rappresentato in eccesso 127
  - L'intervallo di rappresentazione è [-127, 128]
  - Se gli 8 bit dell'esponente contengono 10100011 = 163_{10}
    - L'esponente vale 163-127=36
  - Se gli 8 bit dell'esponente contengono 00100111 = 39_{10}
    - L'esponente vale 39-127=-88

IEEE 754

- La mantissa è sempre nella forma:

1.XXXXXXXXX…X

- Un numero X in virgola mobile secondi IEE 754 viene rappresentato come

$$X = (-1)^S * (1+0.M) * 2^{E-127}$$

IEEE 754

- Quale numero in singola precisione rappresentano i seguenti 32 bit?

1 10000001 01000000000000000000000

- Segno negativo (-)
- Esponente e = 2^7+2^0 -127 = 129-127 = 2
- Mantissa m = $1+2^{-2}$ =1.25

Quindi il numero rappresentato è -1.25 x 2^2 = -5

IEEE 754

- Quale è la rappresentazione a singola precisione del numero 8.5
- Segno positivo (0)
- 8.5 in binario è $1000.1 \cdot 2^0 = 1.0001 \cdot 2^3$
- Esponente e: 3 + 127 = 130 = 10000010
- Mantissa m: 00010000000000000000000

0 10000010 00010000000000000000000

Perchè non rappresento anche gli interi negative usando IEEE 754?

- **CA2 (interi a 32 bit)**: può rappresentare tutti i numeri interi da -2^{31} a $2^{31}-1$.
- **IEEE 754** (single precision, 32 bit): può rappresentare tutti gli interi consecutivi da 0 fino a $2^{24}-1$.
  - Dopo 2^{24}, i numeri interi non possono essere rappresentati esattamente perché la mantissa ha solo 23 bit (più 1 implicito).
  - Quindi, usare IEEE 754 per interi grandi comporterebbe perdita di precisione.

$$2^{24} = \boxed{16.777.216}$$

Cosa succede con 16.777.217?

Perchè non rappresento anche gli interi negative usando IEEE 754?

$$2^{24} = \boxed{16.777.216}$$

Cosa succede con 16.777.217?

Segno (S)	1 bit	0 (positivo)
Esponente (E)	8 bit	$24 + 127 = 151 \Rightarrow$ `10010111`
Mantissa (M)	23 bit	00000000000000000000001

- i 23 bit non bastano a rappresentare quel "1" in fondo con precisione. L'arrotondamento fa sparire l'ultimo bit.

16.777.217 viene quindi approssimato a **16.777.218**

Errore assoluto ed errore relativo

- Rappresentando un numero reale n in virgola mobile si commette un errore di approssimazione.
- In realtà viene rappresentato un numero razionale n' con un numero limitato di cifre significative:

  ERRORE ASSOLUTO: $e_A = n-n'$

  ERRORE RELATIVO: $e_R = e_A / n = (n-n')/n$

- L'ordine di grandezza dell'errore assoluto dipende dal *numero di cifre significative e dall'ordine di grandezza del numero*
- L'ordine di grandezza dell'errore relativo dipende solo dal *numero di cifre significative*

Rappresentazione di altre informazioni

- Caratteri
- Suoni
- Video
- ….

Rappresentazione di caratteri

- Possiamo associare a ogni **carattere** (quale lettera minuscola, lettera maiuscola, vocale accentata e segno di interpunzione) un numero.
- I caratteri possono essere rappresentati in:
  - **ASCII standard**: 1 carattere è rappresentato con 7 bit per un totale di 128 simboli rappresentabili (quali cifre, lettere maiuscole e lettere minuscole);
  - **ASCII estesa**: 1 carattere è rappresentato con 8 bit rappresentabili fino a 256 simboli (i caratteri in più sono usati per esempio per caratteri accentati);
  - **UNICODE**: 1 carattere è rappresentato con un numero maggiore di bit (tra 8 e 32 bit per carattere).

ASCII Standard

- ASCII standard contiene:
  - 26 + 26 lettere (maiuscole + minuscole)
  - 10 cifre decimali (da 0 a 9)
  - segni di interpunzione
  - caratteri di controllo
- Le cifre sono ordinate per valore
- Le lettere maiuscole sono ordinate alfabeticamente
- Le lettere minuscole sono ordinate alfabeticamente (e sono a distanza fissa dalle maiuscole)

ASCII Standard

- Dal 0 a 31 sono dei caratteri di controllo per periferiche
- Da 32 a 47 vari caratteri
- da 48 a 57 cifre decimali
- Da 58 a 64 vari caratteri
- Da 65 a 90 lettere maiuscole dell'alfabeto
- Da 91 a 96 vari caratteri
- Da 97 a 122 lettere minuscole dell'alfabeto
- Da 123 a 127 vari caratteri

ASCII Standard

bit		000	001	010	011	100	101	110	111
	esad.	0	1	2	3	4	5	6	7
0000	0	NUL	DLE	spz	0	@	P	`	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB		7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SS	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	k	[	k	{
1100	C	FF	FS	,	<	L	\	l	|
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SOH	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O		o	DEL

ρ(A) = 100 0001

→ 65

65 = 1 x 2^0 + 1 x 2^6

ρ({) = 111 1011

123=1 x 2^0+ 1 x 2^1 + 1 x 2^3 + 1 x 2^4 + 1 x 2^5 + 1 x 2^6 → 123

La conversione da b=2 a b=10 va da dx verso sx.

ASCII Standard

- Le parole sono sequenze di caratteri.

01101001 i	01101110 n	01100110 f	01101111 o	01110010 r
01101101 m	01100001 a	01110100 t	01101001 i	01100011 c
01100001 a				

bit		000	001	010	011	100	101	110	111
	esad.	0	1	2	3	4	5	6	7
0000	0	NUL	DLE	spz	0	@	P	`	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB		7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SS	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	k	[	k	{
1100	C	FF	FS	,	<	L	\	l	|
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SOH	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O		o	DEL

ASCII Esteso

- Con 1 byte (2^8=256) è possibile realizzare 256 diverse combinazioni

ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

ASCII Esteso

- La tabella ASCII estesa varia come già accennato in base alla zona geografica di utilizzo e al software utilizzato. Le principali estensioni previste dall'ISO 8859 sono:
  - ISO-8859-1(Latin-1), utilizzato nella Zona Europea Occidentale
  - ISO-8859-2 (Latin-2), utilizzato zona Europea Orientale (Serbia, Albania, Ungheria, Romania)
  - ISO-8859-3 (Latin-3), utilizzato nell' Europea del Sud (Malta), include l'Esperanto
  - ISO-8859-4 (Latin-4), obsoleto
  - ISO-8859-5 (Part 5, Cyrillic), alfabeto Cirillico
  - ISO-8859-6 (Part 6, Arabic), alfabeto Arabo
  - ISO-8859-7 (Part 7, Greek), alfabeto Greco
  - ISO-8859-8 (Part 8, Hebrew), alfabeto Ebraico

Osservazioni

- ASCII è un codice accettato da tutti i computer.
  - Usato dai tempi delle telescriventi durante la prima guerra modiale.
- Tuttavia non considera i caratteri internazionali di numerose lingue straniere.
- Per ovviare a tale problematica, è stata introdotta un'ulteriore codifica, ossia UNICODE

UNICODE

- E’ una evoluzione dello standard ASCII
- Unicode è uno standard per la rappresentazione di caratteri
- Codifica tutti i caratteri utilizzati nelle principali lingue del mondo
- Indipendente dalla lingua, dal sistema operativo e dal programma utilizzato
- Inizialmente rappresentato come una codifica su 16 bit, ma poi esteso a 24 e 32 bit
  - Disporre di 32 bit significa avere 4 miliardi di caratteri diversi codificabili!
- Unicode è in continua evoluzione e continua ad aggiungere sempre più caratteri

UNICODE

- Un carattere UNICODE è caratterizzato dal suo codice numerico, detto code point, solitamente rappresentato con 8 cifre esadecimali
  - Esempio: «fi» è rappresentato dal codice 0000FB01
  - Esempio: il simbolo “do doppio diesis strumentale” della notazione musicale greca antica (simile a una lambda maiuscola con una gambetta) è 0001D235
- Con UNICODE, è possibile creare e gestire senza troppa pena documenti multilingue:

A,Δ,Й,ק,م, あ, 叶, 葉

- In particolare, tutti gli standard W3C (incluso HTML) supportano UNICODE
- A marzo 2024 è stato presentato l’ultima versione UNICODE 16.0

Il problema della codifica

- UNICODE può codificare 4.294.967.296 caratteri distinti
- Ogni carattere occupa 32 bit (contro gli 8 delle altre codifiche); i documenti richiedono quindi 4 volte lo spazio
- La quasi totalità dei documenti usa da 60 a 1000 caratteri, per cui basterebbero da 6 a 10 bit.
- Per ovviare a questo problema, e garantire maggiore compatibilità con S.O. e applicazioni che non sono in grado di gestire 32 bit per carattere, UNICODE definisce vari formati di codifica più compatti

UTF-8

- UTF-8 (8-bit UCS/Unicode Transformation Format) è una codifica a lunghezza variabile fra una sequenza di valori a 8 bit e una sequenza di caratteri UNICODE
  - I primi 128 caratteri di UNICODE (0-7F), equivalenti ai caratteri ASCII, sono codificati con il loro codice "naturale"
  - Tutti gli altri caratteri sono codificati con due, tre o quattro valori a 8 bit (byte)

UTF-8

- Nel linguaggio di programmazione Java (e derivati), le stringhe sono codificate con UTF-8; i programmi Java sono quindi in grado di gestire nativamente UNICODE.
- I file system Macintosh, DVD, e alcuni su UNIX usano UTF-8 per i nomi dei file.
- Gli standard relativi al Web e alla e-mail richiedono che un programma compatibile supporti *almeno* UTF-8 come standard di codifica.
- I programmi che trattano testi ASCII sono generalmente UTF-8 compatibili.

Link di interesse

- Il sito principale relativo a UNICODE è http://www.unicode.org
  - la pagina https://symbl.cc/en/unicode/table/ è particolarmente affascinante
- Ultime emoji codificate in unicode (17.0)