



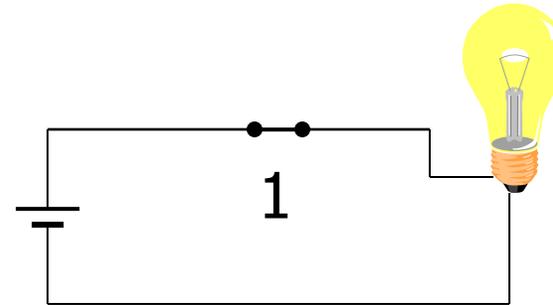
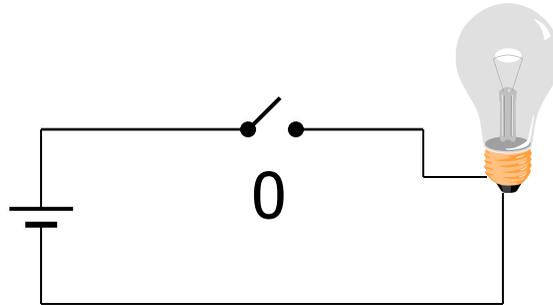
L'algebra di Boole

# Algebra Booleana

- Contempla due costanti **0** e **1** (**falso** e **vero**)
- Corrispondono a due stati che si escludono a vicenda
- Possono descrivere lo stato di apertura o chiusura di un generico contatto o di un circuito a più contatti

- Si **AND, OR, NOT** definiscono gli operatori fondamentali delle operazioni

fra i valori booleani:



# L'operazione di OR

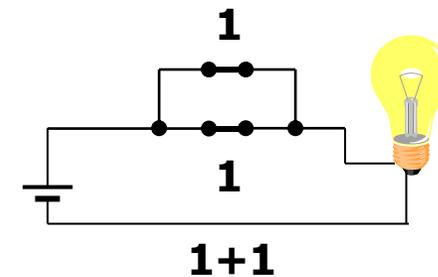
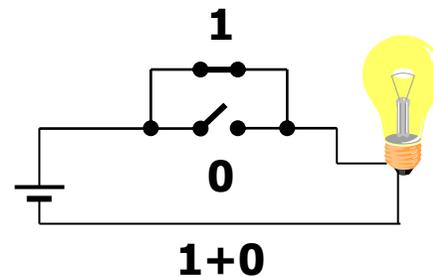
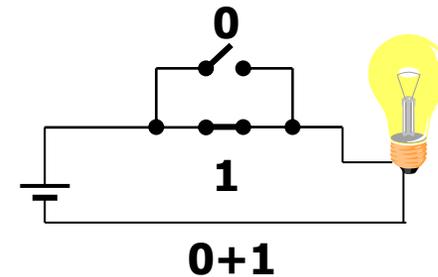
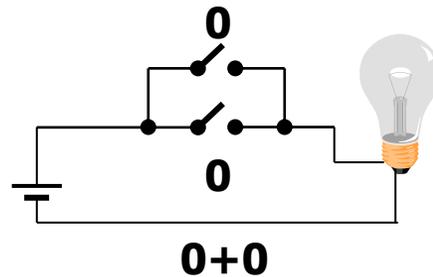
- Si definisce l'operazione di **somma logica** (OR): il valore della somma logica è il simbolo 1 se il valore di almeno uno degli operandi è il simbolo 1

$$0+0 = 0$$

$$0+1 = 1$$

$$1+0 = 1$$

$$1+1 = 1$$



# L'operazione di AND

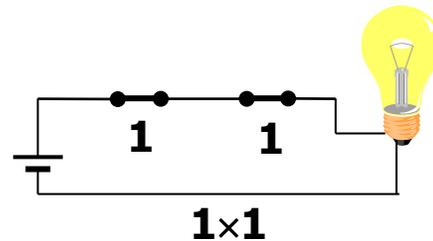
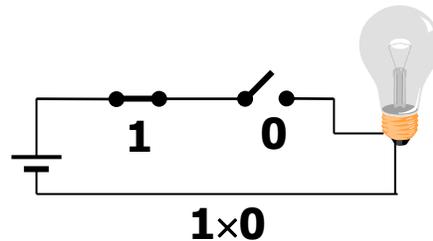
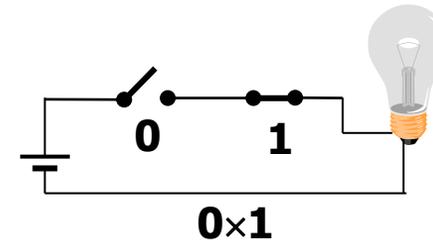
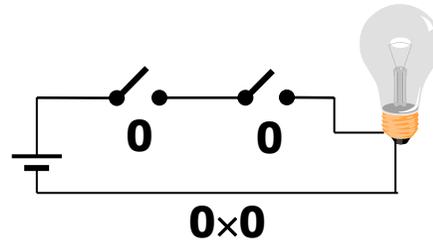
- Si definisce l'operazione di **prodotto logico** (AND): il valore del prodotto logico è il simbolo 1 se il valore di tutti gli operandi è il simbolo 1

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$



# La negazione NOT

- Si definisce l'operatore di **negazione** (NOT):  
l'operatore inverte il valore della costante su cui opera

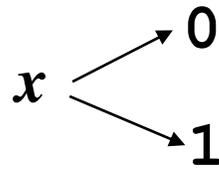
$$\begin{aligned}\bar{0} &= 1 \\ \bar{1} &= 0\end{aligned}$$

- Dalla definizione...

$$\begin{aligned}\overline{\bar{0}} &= 0 \\ \overline{\bar{1}} &= 1\end{aligned}$$

# Variabili binarie

- Una variabile binaria indipendente può assumere uno dei due valori **0** e **1**



- Date  $n$  variabili binarie indipendenti, la loro somma logica (OR) è

$$x_1 + x_2 + \dots + x_n = \begin{cases} \mathbf{1} & \text{se almeno una } x_i \text{ vale } \mathbf{1} \\ \mathbf{0} & \text{se } x_1 = x_2 = \dots = x_n = \mathbf{0} \end{cases}$$

# AND e NOT con variabili binarie

- Date  $n$  variabili binarie indipendenti, il loro prodotto logico (AND) è

- La negazione di una variabile  $x$  è

$$x_1 \times x_2 \times \dots \times x_n = \begin{cases} 0 & \text{se almeno una } x_i \text{ vale } 0 \\ 1 & \text{se } x_1 = x_2 = \dots = x_n = 1 \end{cases}$$

$$\bar{x} = 0 \quad \text{se} \quad x = 1$$

$$\bar{x} = 1 \quad \text{se} \quad x = 0$$

# Alcune identità

- Si verificano

- Ad esempio...

$$x + 1 = 1$$

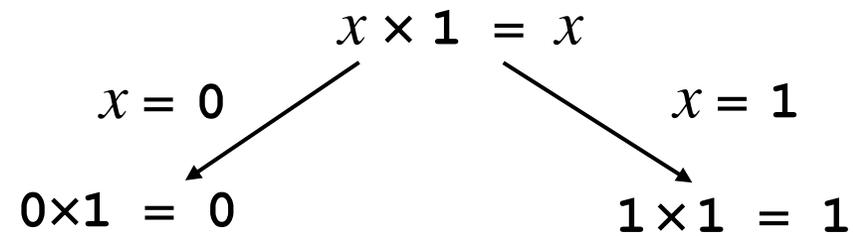
$$x \times 1 = x$$

$$x + 0 = x$$

$$x \times 0 = 0$$

$$x + x = x$$

$$x \times x = x$$



OK!

# I numeri nell'antica Roma – 2

- La scrittura dei numeri avveniva combinando additivamente i segni
- Per agevolare scrittura e lettura si diffuse più tardi un sistema sottrattivo già utilizzato, ad esempio, dagli Assiri (che ha traccia anche nelle forme verbali come ad esempio “undeviginti”, stessa cosa di “decem et novem”)
  - Un simbolo posto alla sinistra di un simbolo di quantità maggiore viene sottratto, così IX e VIIII indicano entrambi il numero 9
- Ancora in epoca tarda, un segno che prese l'aspetto di una linea orizzontale posta sopra le lettere serviva per indicarne la moltiplicazione per 1000

# Dall'India... il sistema decimale – 1

- La civiltà indiana, più antica delle civiltà classiche, è già documentata dal 3000 a.C.
- Sebbene l'uso della matematica dovesse essere ben sviluppato già in epoca arcaica, i primi testi che ci sono giunti risalgono al V secolo d.C.
- Non è però ancora chiaro dove e quando si sia sviluppato il sistema di notazione decimale posizionale che, in seguito, attraverso gli **Arabi**, si è diffuso in Europa

# Dall'India... il sistema decimale – 2

- Tale sistema viene utilizzato nell'opera del matematico indiano vissuto attorno al 500 d.C. Aryabhata, la più antica che ci è pervenuta se si eccettuano frammenti sparsi di matematici anteriori, dove però manca ancora l'uso di un simbolo zero
- Testimonianze di scritture in forma posizionale si registrano anche prima del manuale di Aryabhata, mentre per avere datazioni sicure di forme complete in cui compare anche il simbolo zero occorre arrivare al IX secolo d.C.

# Dall'India... il sistema decimale – 3

- L'idea di usare un numero limitato di simboli a cui dare valore diverso a seconda della posizione occupata può essere stata, secondo alcuni studiosi, sviluppata dagli **Indiani** per conoscenza diretta — o ereditata dai Greci — del sistema sessagesimale babilonese
- Gli Indiani avrebbero allora iniziato ad utilizzare solamente i primi 9 simboli del loro sistema decimale in caratteri *Brahmi*, in uso dal III secolo a.C.

# Dall'India... il sistema decimale – 4

- I simboli assumono forme diverse a seconda delle zone e dei periodi, ma sono comunque quelli che gli Arabi più tardi utilizzarono e che, dalla forma araba, sono passati in Europa, fino alla forma definitiva resa uniforme dalla stampa nel XV secolo

1	2	3	4	5	6	7	8	9	0
१	२	३	४	५	६	७	८	९	०

# Sistemi di numerazione posizionali

- Sistemi di numerazione **posizionali**:

La **base** del sistema di numerazione

Le **cifre** del sistema di numerazione

Il numero è scritto specificando le cifre in ordine ed il suo valore dipende dalla **posizione relativa** delle cifre

**Esempio**: Il sistema decimale (Base 10)

Cifre : 0 1 2 3 4 5 6 7 8 9

$$\begin{array}{cccc} & 5 & 6 & 4 & 1 \\ & \uparrow & \uparrow & \uparrow & \uparrow \\ \text{Posizione:} & 3 & 2 & 1 & 0 \end{array} \quad 5641 = 5 \cdot 10^3 + 6 \cdot 10^2 + 4 \cdot 10^1 + 1 \cdot 10^0$$

# Il sistema binario (B=2)

- La base 2 è la più piccola per un sistema di numerazione

Cifre: 0 1 – **bit** (binary digit)

Esempi:

$$(101101)_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 32 + 0 + 8 + 4 + 0 + 1 = (45)_{10}$$

Forma polinomiale

$$(0,0101)_2 = 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = 0 + 0,25 + 0 + 0,0625 = (0,3125)_{10}$$

$$(11,101)_2 = 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 2 + 1 + 0,5 + 0 + 0,125 = (3,625)_{10}$$

# Dal bit al byte

- Un **byte** è un insieme di 8 bit (un numero binario a 8 cifre)

$b_7b_6b_5b_4b_3b_2b_1b_0$

- Con un byte si rappresentano i numeri interi fra 0 e  $2^8-1 = 255$

00000000

00000001

00000010

00000011

.....

11111110

11111111

$2^8 = 256$  valori distinti

- È l'elemento base con cui si rappresentano i dati nei calcolatori
- Si utilizzano sempre dimensioni multiple (di potenze del 2) del byte: 2 byte (16 bit), 4 byte (32 bit), 8 byte (64 bit)...

# Dal byte al kilobyte

- Potenze del 2

$$2^4 = 16$$

$$2^8 = 256$$

$$2^{16} = 65536$$

- Cosa sono KB (Kilobyte), MB (Megabyte), GB (Gigabyte)?

$$2^{10} = 1024 \quad (\text{K=Kilo})$$

$$2^{20} = 1048576 \quad (\text{M=Mega})$$

$$2^{30} = 1073741824 \quad (\text{G=Giga})$$

$$1 \text{ KB} = 2^{10} \text{ byte} = 1024 \text{ byte}$$

$$1 \text{ MB} = 2^{20} \text{ byte} = 1048576 \text{ byte}$$

$$1 \text{ GB} = 2^{30} \text{ byte} = 1073741824 \text{ byte}$$

$$1 \text{ TB} = 2^{40} \text{ byte} = 1099511627776 \text{ byte (Terabyte)}$$

# La rappresentazione dei dati e l'aritmetica degli elaboratori

# Numeri interi positivi

- I numeri interi positivi sono rappresentati all'interno dell'elaboratore utilizzando un multiplo del byte (generalmente 4/8 byte)
- Se l'intero si rappresenta con un numero di cifre minore, vengono aggiunti zeri nelle cifre più significative

**Esempio:** 12 viene rappresentato in un byte come...

00001100

# Numeri con segno

- Per rappresentare numeri con segno, occorre utilizzare un bit per definire il segno del numero
- Si possono usare tre tecniche di codifica

- Modulo e segno
- Complemento a 2
- Complemento a 1

# Modulo e segno

- Il bit più significativo rappresenta il segno: 0 per i numeri positivi, 1 per quelli negativi
- Esiste uno zero positivo (00...0) e uno zero negativo (10...0)
- Se si utilizzano n bit si rappresentano tutti i numeri compresi fra  $-(2^{n-1}-1)$  e  $+2^{n-1}-1$

**Esempio:** con 4 bit si rappresentano i numeri fra  $-7$  ( $-(2^3-1)$ ) e  $+7$  ( $2^3-1$ )

0000	+0	1000	-0
0001	+1	1001	-1
0010	+2	1010	-2
0011	+3	1011	-3
0100	+4	1100	-4
0101	+5	1101	-5
0110	+6	1110	-6
0111	+7	1111	-7

positivi                      negativi

# Complemento a 2

- Il complemento a 2 di un numero binario  $(N)_2$  a  $n$  cifre è il numero

$$2^n - (N)_2 = \underbrace{10\dots\dots 0}_n - (N)_2$$

- Il complemento a 2 si calcola...

- ↳ Effettuando il complemento a 1 del numero di partenza (negazione di ogni cifra): si trasforma ogni 0 in 1 e ogni 1 in 0
- ↳ Aggiungendo 1 al numero ottenuto

- Oppure: a partire da destra, lasciando invariate tutte le cifre fino al primo 1 compreso, quindi invertendo il valore delle rimanenti

01010111		10000000	$2^8$
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓		01111111	$2^8 - 1$
10101000	← complemento a 1	01010111	$N$
		-----	
		10101000	$2^8 - 1 - N$
	+ 1		
-----		10101001	<del><math>2^8 - 1 - N + 1</math></del>
10101001			

# Interi in complemento a 2

- I **numeri positivi** sono rappresentati (come) in modulo e segno
- I **numeri negativi** sono rappresentati in complemento a 2  $\Rightarrow$  la cifra più significativa ha sempre valore 1
- Lo zero è rappresentato come numero positivo (con una sequenza di n zeri)
- Il campo dei numeri rappresentabili varia da  $-2^{n-1}$  a  $+2^{n-1}-1$

**Esempio:** numeri a 4 cifre

0000	+0	1000	-8
0001	+1	1001	-7
0010	+2	1010	-6
0011	+3	1011	-5
0100	+4	1100	-4
0101	+5	1101	-3
0110	+6	1110	-2
0111	+7	1111	-1

Nota: 0111 +7  
1000 -8

# Interi a 16 bit

- Numeri interi rappresentati su 16 bit in complemento a 2:

Il numero intero positivo più grande è  $2^{15}-1=(32767)_{10}$

0111 1111 1111 1111  
0x 7 F F F

Il numero intero negativo più piccolo è  $-2^{15}=(-32768)_{10}$

1000 0000 0000 0000 → 0111 1111 1111 1111 +  
0x 8 0 0 0 1  
1000 0000 0000 0000

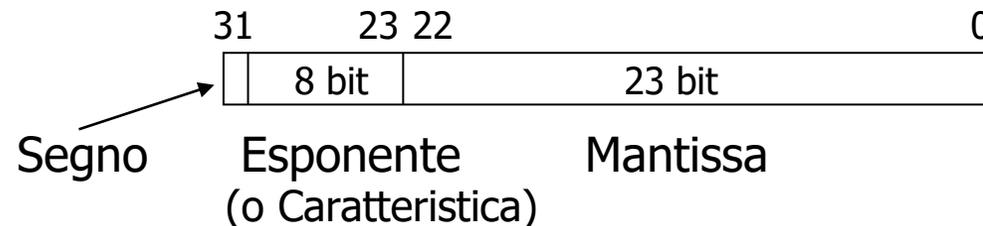
Il numero intero  $-1$  è rappresentato come

1111 1111 1111 1111 → 0000 0000 0000 0000 +  
0x F F F F 1  
0000 0000 0000 0001

# Numeri in virgola mobile

- La rappresentazione dei numeri in virgola mobile è in relazione con la **notazione scientifica** (es.  $1.2 \times 10^2 = 120$ )
- La IEEE ha previsto uno standard per la rappresentazione in virgola mobile
  - **singola precisione** (32 bit = 4 byte)
  - **doppia precisione** (64 bit = 8 byte)
  - **quadrupla precisione** (128 bit = 16 byte)

Singola precisione



Il valore è

$$\begin{aligned} & (-1)^S 1.M \times 2^{E-127} && \text{se } E \neq 0 \\ & (-1)^S 0.M \times 2^{-126} && \text{se } E = 0 \end{aligned}$$

**Eccesso:** vale  $2^{t-1}-1$ , se  $t$  è il numero di cifre riservate alla caratteristica → rappresentazione "interna" dell'esponente sempre positiva



# L'aritmetica floating-point: addizione

## # Metodo per il calcolo dell'addizione

1. Se le caratteristiche dei numeri sono diverse, si considera il numero con caratteristica minore e...
  - 1.1 Si trasla la mantissa di un posto a destra
  - 1.2 Si incrementa la caratteristica di 1, fino a quando le due caratteristiche sono uguali, e corrispondono alla caratteristica del risultato
2. La mantissa del risultato è ottenuta dalla somma delle due mantisse
3. Se l'addizione comporta un riporto oltre la cifra più significativa, si trasla la mantissa del risultato a destra di un posto, il riporto nel bit più significativo, e si incrementa la caratteristica di 1

# L'aritmetica floating-point: moltiplicazione

## # Metodo per il calcolo della moltiplicazione

1. Si moltiplicano le due mantisse
2. Si addizionano le due caratteristiche
3. Si trasla a sinistra il prodotto delle due mantisse fino ad ottenere un 1 come cifra più significativa; si diminuisce la caratteristica di 1 per ogni traslazione eseguita
4. Si tronca la mantissa al numero di bit utilizzati nella rappresentazione; la mantissa del prodotto è il risultato del troncamento
5. Si sottrae l'eccesso alla somma delle caratteristiche, ottenendo la caratteristica del prodotto

# L'aritmetica degli elaboratori – 1

- L'aritmetica "interna" degli elaboratori differisce notevolmente dall'aritmetica classica
- Sebbene le stesse operazioni possano essere realizzate secondo modalità diverse su elaboratori diversi, si riscontrano alcune caratteristiche comuni:
  - Rappresentazione binaria dei numeri
  - Rango finito dei numeri rappresentabili
  - Precisione finita dei numeri
  - Operazioni espresse in termini di operazioni più semplici

# L'aritmetica degli elaboratori – 2

- **Rango finito dei numeri rappresentabili**
  - Qualunque sia la codifica utilizzata, esistono sempre il più grande ed il più piccolo numero rappresentabile
  - I limiti inferiore e superiore del rango di rappresentazione dipendono sia dal tipo di codifica, sia dal numero di bit utilizzati
  - Se il risultato di un'operazione non appartiene al rango dei numeri rappresentabili, si dice che si è verificato un overflow (un **underflow**, più precisamente, se il risultato è più piccolo del più piccolo numero rappresentabile)

# L'aritmetica degli elaboratori – 3

- Precisione finita dei numeri

- La **precisione** della rappresentazione di un numero frazionario è una misura di quanto essa corrisponda al numero che deve essere rappresentato
- Negli elaboratori, i numeri frazionari sono rappresentati in virgola mobile (**floating-point**), utilizzando un numero finito di bit
- È plausibile che un numero reale non ammetta una rappresentazione finita, quindi dovrà essere codificato in maniera approssimata
- Negli elaboratori si rappresentano soltanto numeri razionali (fino ad una data precisione)

# L'aritmetica degli elaboratori – 4

- Operazioni espresse in termini di operazioni più semplici
  - ✦ La maggior parte degli elaboratori non possiede circuiti in grado di eseguire direttamente tutte le operazioni:
    - ◆ La sottrazione si realizza per mezzo di una complementazione e di un'addizione
    - ◆ La moltiplicazione si realizza per mezzo di una successione di addizioni e di **shift** (traslazioni)
    - ◆ La divisione si realizza per mezzo di una successione di shift e sottrazioni
  - ✦ Le operazioni più semplici sono eseguite direttamente da appositi circuiti (in **hardware**); le operazioni più complesse sono realizzate mediante l'esecuzione di successioni di operazioni più semplici, sotto il controllo di programmi appositamente realizzati, e generalmente memorizzati permanentemente (in **firmware**)

# Codifica dei caratteri alfabetici – 1

- Oltre ai numeri, molte applicazioni informatiche elaborano caratteri (simboli)
- Gli elaboratori elettronici trattano numeri
- Si codificano i caratteri e i simboli per mezzo di numeri
- Per poter scambiare dati (testi) in modo corretto, occorre definire uno standard di codifica

A	—————>	01000001
3	—————>	00110011
\$	—————>	00100100

# Codifica dei caratteri alfabetici – 2

- Quando si scambiano dati, deve essere noto il tipo di codifica utilizzato
- La codifica deve prevedere le lettere dell'alfabeto, le cifre numeriche, i simboli, la punteggiatura, i caratteri speciali per certe lingue (æ, ã, ë, è,...)
- Lo standard di codifica più diffuso è il **codice ASCII**, per **American Standard Code for Information Interchange**

# Codifica ASCII

- Definisce una tabella di corrispondenza fra ciascun carattere e un codice a **7 bit** (128 caratteri)
- I caratteri, in genere, sono rappresentati con **1 byte** (8 bit); i caratteri con il bit più significativo a 1 (quelli con codice dal 128 al 255) rappresentano un'estensione della codifica
- La tabella comprende sia **caratteri di controllo** (codici da 0 a 31) che **caratteri stampabili**
- I caratteri alfabetici/numerici hanno codici ordinati secondo l'ordine alfabetico/numerico

0 48	A 65	a 97
1 49	B 66	b 98
.....	.....	.....
8 56	Y 89	y 121
9 57	Z 90	z 122
cifre	maiuscole	minuscole

# Caratteri di controllo ASCII

- I caratteri di controllo (codice da 0 a 31) hanno funzioni speciali
- Si ottengono o con tasti specifici o con una sequenza **Ctrl+carattere**

<b>Ctrl</b>	<b>Dec</b>	<b>Hex</b>	<b>Code</b>	<b>Nota</b>
^@	0	0	NULL	carattere nullo
^A	1	1	SOH	partenza blocco
.....	...	...	.....	.....
^G	7	7	BEL	beep
^H	8	8	BS	backspace
^I	9	9	HT	tabulazione orizzontale
^J	10	A	LF	line feed (cambio linea)
^K	11	B	VT	tabulazione verticale
^L	12	C	FF	form feed (alim. carta)
^M	13	D	CR	carriage return (a capo)
.....	...	...	.....	.....
^Z	26	1A	EOF	fine file
^[	27	1 B	ESC	escape
.....	...	...	.....	.....
^_	31	1F	US	separatore di unità

# Caratteri ASCII stampabili

Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr
32	20	SPACE	48	30	0	64	40	@	80	50	P	96	60	`	112	70	p
33	21	!	49	31	1	65	41	A	81	51	Q	97	61	a	113	71	q
34	22	"	50	32	2	66	42	B	82	52	R	98	62	b	114	72	r
35	23	#	51	33	3	67	43	C	83	53	S	99	63	c	115	73	s
36	24	\$	52	34	4	68	44	D	84	54	T	100	64	d	116	74	t
37	25	%	53	35	5	69	45	E	85	55	U	101	65	e	117	75	u
38	26	&	54	36	6	70	46	F	86	56	V	102	66	f	118	76	v
39	27	'	55	37	7	71	47	G	87	57	W	103	67	g	119	77	w
40	28	(	56	38	8	72	48	H	88	58	X	104	68	h	120	78	x
41	29	)	57	39	9	73	49	I	89	59	Y	105	69	i	121	79	y
42	2A	*	58	3A	:	74	4A	J	90	5A	Z	106	6A	j	122	7A	z
43	2B	+	59	3B	;	75	4B	K	91	5B	[	107	6B	k	123	7B	{
44	2C	,	60	3C	<	76	4C	L	92	5C	\	108	6C	l	124	7C	
45	2D	-	61	3D	=	77	4D	M	93	5D	]	109	6D	m	125	7D	}
46	2E	.	62	3E	>	78	4E	N	94	5E	^	110	6E	n	126	7E	~
47	2F	/	63	3F	?	79	4F	O	95	5F	_	111	6F	o	127	7F	DEL

**Nota:** il valore numerico di una cifra può essere calcolato come differenza del suo codice ASCII rispetto al codice ASCII della cifra 0 (es. '5'-'0' = 53-48 = 5)

# Tabella ASCII estesa

- I codici oltre il 127 non sono compresi nello standard originario

128	Ç	144	É	160	á	176	☐	193	⊥	209	⌘	225	β	241	±
129	ü	145	æ	161	í	177	☐	194	⊥	210	⌘	226	Γ	242	≥
130	é	146	Æ	162	ó	178	☐	195	⊥	211	⌘	227	π	243	≤
131	â	147	ô	163	ú	179		196	—	212	⌘	228	Σ	244	∫
132	ä	148	ö	164	ñ	180	⊥	197	+	213	⌘	229	σ	245	∫
133	à	149	ò	165	Ñ	181	⊥	198	⊥	214	⌘	230	μ	246	÷
134	â	150	û	166	ª	182	⊥	199	⊥	215	⌘	231	τ	247	≈
135	ç	151	ù	167	º	183	⌘	200	⌘	216	⊥	232	Φ	248	°
136	ê	152	_	168	¿	184	⌘	201	⌘	217	⊥	233	⊕	249	·
137	ë	153	Ö	169	_	185	⊥	202	⌘	218	⌘	234	Ω	250	·
138	è	154	Û	170	¬	186	⊥	203	⌘	219	■	235	δ	251	√
139	í	156	£	171	½	187	⌘	204	⊥	220	■	236	∞	252	_
140	î	157	¥	172	¼	188	⌘	205	=	221	■	237	φ	253	²
141	ï	158	_	173	¡	189	⌘	206	⊥	222	■	238	ε	254	■
142	Ä	159	f	174	«	190	⊥	207	⊥	223	■	239	∧	255	
143	Å	192	⌘	175	»	191	⌘	208	⌘	224	α	240	≡		