

Rappresentazione digitale dell'informazione

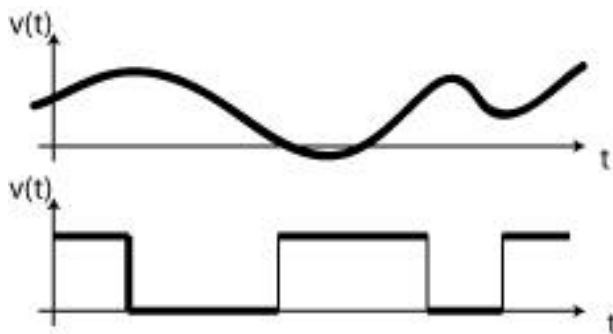
Indice

- L'aritmetica dei calcolatori
- Numeri a precisione finita
- Sistemi di numerazione posizionali
- I sistemi di numerazione a base non decimale
- Il sistema di numerazione binario
- Il formato floating-point

- Rappresentazione di caratteri
- Rappresentazione di suoni
- Rappresentazione di immagini

I sistemi digitali

Segnale analogico: Un segnale è analogico quando i valori utili che lo rappresentano sono continui (infiniti) in un intervallo e non numerabili.



Segnale digitale: Un segnale è digitale quando i valori utili che lo rappresentano sono discreti e finiti.

I calcolatori moderni utilizzano due stati logici (binari), ma è possibile progettare sistemi digitali con logica a più stati (detta anche logica multi valori).

L'aritmetica dei calcolatori

L'aritmetica usata dai calcolatori è diversa da quella comunemente utilizzata dalle persone

La **precisione** con cui i numeri possono essere espressi è finita e predeterminata poiché questi devono essere memorizzati entro un limitato spazio di memoria

$$\sqrt{2} = \begin{array}{|c|c|c|c|c|c|c|} \hline 1 & . & 4 & 1 & 4 & 2 & 1 \\ \hline \end{array} \quad 3 \quad 5 \quad 6 \quad 2$$

La **rappresentazione** è normalmente ottenuta utilizzando il sistema binario poiché più adatto a essere maneggiato dal calcolatore

$$124 \Rightarrow 01111100$$

Numeri a precisione finita (1)

I **numeri a precisione finita** sono quelli rappresentati con un numero finito di cifre.

Fissate le caratteristiche del numero è determinato anche l'insieme di valori rappresentabili

Esempio: *Numeri non rappresentabili con 3 cifre senza virgola e senza segno*

Numeri superiori a 999

Numeri negativi

Frazioni

1	5	9
---	---	---

Le operazioni con i numeri a precisione finita causano errori ogniqualvolta il loro risultato non appartiene all'insieme dei valori rappresentabili:

- **Underflow:** si verifica quando il risultato dell'operazione è minore del più piccolo valore rappresentabile
- **Overflow:** si verifica quando il risultato dell'operazione è maggiore del più grande valore rappresentabile
- **Non appartenenza all'insieme:** si verifica quando il risultato dell'operazione, pur non essendo troppo grande o troppo piccolo, non appartiene all'insieme dei valori rappresentabili

Esempio: *Numeri a precisione finita con 3 cifre senza virgola e senza segno*

$$600+600 = 1200 \Rightarrow \text{Overflow}$$

$$300-600 = -300 \Rightarrow \text{Underflow}$$

$$007/002 = 3.5 \Rightarrow \text{Non appartenenza all'insieme}$$

Numeri a precisione finita (2)

Si noti che, a differenza dei numeri interi, i numeri a precisione finita non rispettano la chiusura rispetto alle operazioni di somma, sottrazione e prodotto.

Esempio: Risultati non rappresentabili con 3 cifre senza virgola e senza segno

Operazioni	Interi	Precisione finita
Somma: $600+600$	1200	Overflow
Sottrazione: $300-600$	-300	Underflow
Prodotto: 050×050	2500	Overflow


Anche l'algebra dei numeri a precisione finita è diversa da quella convenzionale. Poiché alcune delle proprietà non vengono rispettate:

Proprietà associativa: $a + (b - c) = (a + b) - c$


Proprietà distributiva: $a \times (b - c) = a \times b - a \times c$

Non sono rispettate poiché in base all'ordine con cui vengono eseguite le operazioni si può verificare o meno un errore

Esempio: Operazioni con numeri a precisione finita di 3 cifre senza virgola e senza segno


$$400 + (300 - 500) = (400 + 300) - 500$$
$$400 + (-200) = 700 - 500$$

Underflow

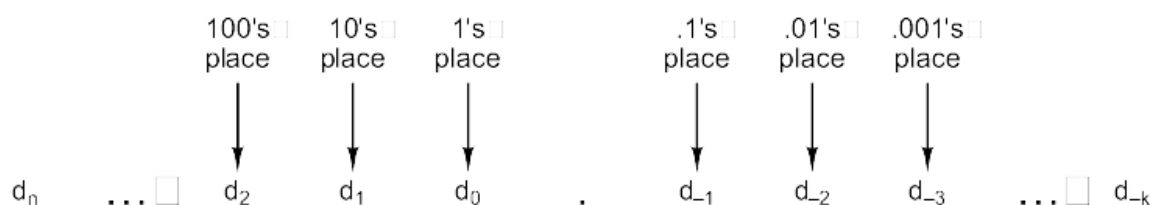
$$50 \times (50 - 40) = 50 \times 50 - 50 \times 40$$

$$50 \times 10 = 2500 - 2000$$

Overflow Overflow

ATTENZIONE: non confondere i numeri negativi con le operazioni di sottrazione

Notazione posizionale (1)

I sistemi di numerazione posizionale associano alle cifre un diverso valore in base alla posizione che occupano nella stringa che compone il numero.



$$\text{Valore} = \sum_{i=-k}^n d_i \times 10^i$$

Esempio: *Rappresentazione posizionale di 5798.46*

$$\begin{aligned} &5 \times 10^3 + 7 \times 10^2 + 9 \times 10^1 + 8 \times 10^0 + 4 \times 10^{-1} + 6 \times 10^{-2} \\ &= 5000 + 700 + 90 + 8 + 0.4 + 0.06 \end{aligned}$$

- Un sistema di numerazione posizionale è definito dalla **base (o radice)** utilizzata per la rappresentazione.
- Un sistema posizionale in base b richiede b simboli per rappresentare i diversi valori tra 0 e $(b-1)$

Sistema decimale ($b=10$) 0 1 2 3 4 5 6 7 8 9

Sistema binario ($b=2$) 0 1: ogni cifra, detta *bit* (**B**inary dig**IT**), può essere rappresentata direttamente tramite un livello elettrico di tensione

Sistema ottale ($b=8$) 0 1 2 3 4 5 6 7

Sistema esadecimale ($b=16$) 0 1 2 3 4 5 6 7 8 9 A B C D E F: è utilizzato nel linguaggio dell'assemblatore poiché è molto compatto e semplice da scandire. Inoltre ben si presta alla traduzione in valori binari poiché ogni cifra corrisponde esattamente a 4 cifre binarie.

Notazione posizionale (2)

A ogni numero corrisponderanno rappresentazioni diverse in basi diverse

Binary	1	1	1	1	1	0	1	0	0	0	1
	$1 \cdot 2^{10} +$	$1 \cdot 2^9 +$	$1 \cdot 2^8 +$	$1 \cdot 2^7 +$	$1 \cdot 2^6 +$	$0 \cdot 2^5 +$	$1 \cdot 2^4 +$	$0 \cdot 2^3 +$	$0 \cdot 2^2 +$	$0 \cdot 2^1 +$	$1 \cdot 2^0$
	1024	+ 512	+ 256	+ 128	+ 64	+ 0	+ 16	+ 0	+ 0	+ 0	+ 1
Octal	3	7	2	1							
	$3 \cdot 8^3 +$	$7 \cdot 8^2 +$	$2 \cdot 8^1 +$	$1 \cdot 8^0$							
	1536	+ 448	+ 16	+ 1							
Decimal	2	0	0	1							
	$2 \cdot 10^3 +$	$0 \cdot 10^2 +$	$0 \cdot 10^1 +$	$1 \cdot 10^0$							
	2000	+ 0	+ 0	+ 1							
Hexadecimal	7	D	1								.
	$7 \cdot 16^2 +$	$13 \cdot 16^1 +$	$1 \cdot 16^0$								
	1792	+ 208	+ 1								

Dato che l'insieme dei simboli utilizzati dalle varie basi non è disgiunto è necessario aggiungere al numero un pedice che indichi la radice utilizzata.

$$11111010001_2 = 3721_8 = 2001_{10} = 7D1_{16}$$

Conversione tra basi (1)

Binario \Leftrightarrow Ottale: dato che una cifra del sistema ottale è rappresentabile esattamente con tre cifre del sistema binario, la conversione può essere ottenuta raggruppando le cifre binarie a 3 a 3 a partire dalla virgola binaria. L'operazione contraria è ugualmente semplice, ogni cifra ottale viene convertita in esattamente tre cifre binarie.

Esadecimale \Leftrightarrow binario: il processo di conversione è equivalente a quello binario-ottale ma le cifre binarie devono essere raggruppate a 4 a 4.

Example 1

Hexadecimal

Binary

Octal

1	9	4	8	.	B	6		
$\overbrace{0001100101001000.101101100}$								
1	4	5	1	0	.	5	5	4

Example 2

Hexadecimal

Binary

Octal

7	B	A	3	.	B	C	4		
$\overbrace{0111101110100011.101111000100}$									
7	5	6	4	3	.	5	7	0	4

Conversione tra basi (2)

Decimale \Rightarrow Binario (Metodo generale): si procede sottraendo al numero da decomporre la più grande potenza di 2 minore del numero da decomporre. Il processo viene applicato ricorsivamente al resto della sottrazione. Il risultato binario si ottiene ponendo a uno le cifre corrispondenti alle potenze che sono state utilizzate nella decomposizione.

$$1492.25 = 2^{10} + 468.25$$

$$468.25 = 2^8 + 212.25$$

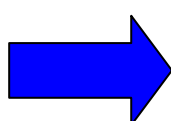
$$212.25 = 2^7 + 84.25$$

$$84.25 = 2^6 + 20.25$$

$$20.25 = 2^4 + 4.25$$

$$4.25 = 2^2 + 0.25$$

$$0.25 = 2^{-2}$$



10111010100.01

Decimale \Rightarrow Binario (Solo per interi): la codifica viene ottenuta direttamente procedendo in modo ricorsivo. Si divide il numero per 2: il resto (0 o 1) farà parte del risultato mentre il quoziente verrà utilizzato come input al seguente passo di ricorsione.

Quotients	Remainders
1492	
746	0
373	0
186	1
93	0
46	1
23	0
11	1
5	1
2	1
1	0
0	1

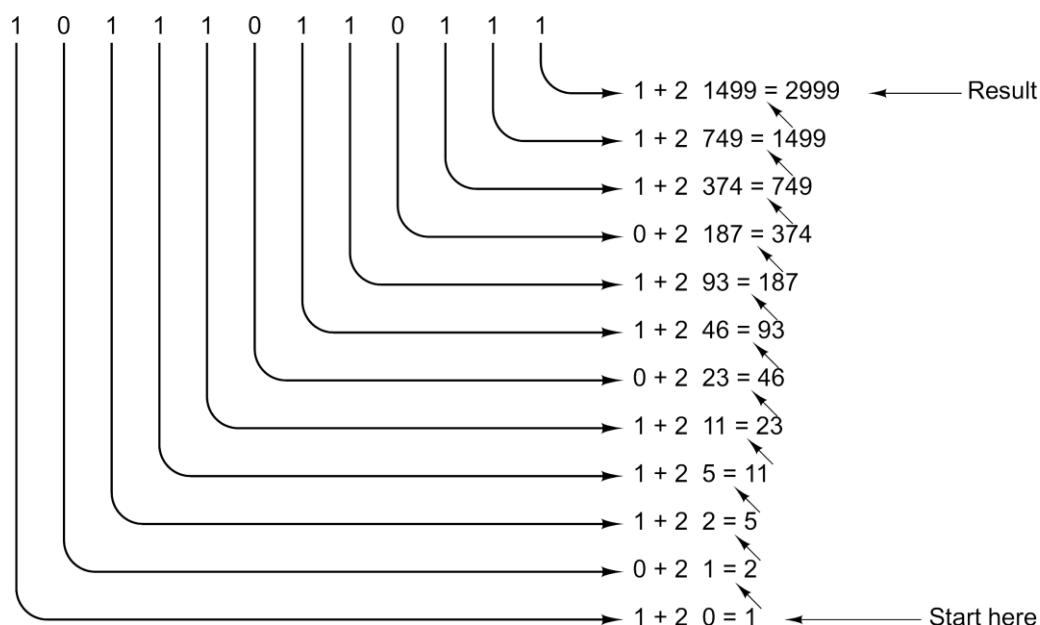
1 0 1 1 1 0 1 0 0 = 1492₁₀

Conversione tra basi (3)

Binario \Rightarrow Decimale (Primo metodo): si procede traducendo le singole cifre binarie alle corrispondenti potenze di due in base decimale e sommando i risultati parziali:

$$10111010100.01$$
$$2^{10} + 2^8 + 2^7 + 2^6 + 2^4 + 2^2 + 2^{-2}$$
$$1024 + 256 + 128 + 64 + 16 + 4 + 0.25$$
$$1492.25$$

Binario \Rightarrow Decimale (Secondo metodo): la codifica viene ottenuta ricreando il valore posizionale di ogni cifra tramite successive moltiplicazioni per due a partire dalle cifre più significative verso quelle meno significative.



Altre conversioni: la conversione da decimale a ottale e da decimale a esadecimale può essere fatta passando per il sistema binario, oppure applicando i metodi precedentemente descritti e ponendo attenzione al corretto uso delle basi.

Esercizi

1. Si convertano i seguenti numeri decimali in base 2:

371 3224 114.65625

2. Si convertano i seguenti numeri binari in base 8 e 16:

11100110100110 1111001100011100

3. Si convertano i seguenti numeri esadecimali in base 2:

FA31C CCCAB001

4. Si convertano i seguenti numeri esadecimali in base 10:

AAB E0CC

Il calcolatore e i numeri binari

Prima di procedere nello studio dei numeri binari è bene ricordare che il codice e i dati di un programma vengono memorizzati, e successivamente, utilizzati da un calcolatore la cui architettura interna stabilisce il loro formato e il campo dei valori che possono assumere.

La più importante unità di misura dell'informazione manipolata dal calcolatore è il **BYTE** composto da 8 bit.

0	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---

Nel byte il bit più a destra è quello meno significativo mentre quello a sinistra è quello più significativo.

Sequenze di bit più lunghe di un byte sono denominate **WORD**. La loro lunghezza dipende dalle caratteristiche del sistema, ma è sempre un multiplo del byte: 16/32/64/128 bit.

L'intervallo di valori codificabili dipende ovviamente dal numero di configurazioni possibili e dal tipo di dato da rappresentare. Con n bit sono possibili 2^n configurazioni.

Esempio: *Intervallo di interi positivi rappresentabili con n bit*

n	Num. Configurazioni	Intervallo
1	2	0 - 1
8	256	0 - 255
16	65.536	0 – 65.535
32	4.294.967.296	0 - 4.294.967.295
64	18.446.744.073.709.551.616	0 – 18.446.744.073.709.551.615

Unità di misura nel sistema binario

Il bit rappresenta la più piccola unità di misura dell'informazione memorizzabile in un calcolatore. I sistemi moderni memorizzano e manipolano miliardi di bit; per questo motivo sono stati definiti molti multipli.

Nome	Sigla	In bit	In byte	In potenze di 2
Bit	Bit	1 bit	1/8	$2^1=2$ stati
Byte	Byte	8	1	$2^8=256$ stati
KiloByte	KB	8.192	1.024	2^{10} byte
MegaByte	MB	8.388.608	1.048.576	2^{20} byte
GigaByte	GB	8.589.934.592	1.073.741.824	2^{30} byte
TeraByte	TB	8.796.093.022.208	1.099.511.627.776	2^{40} byte

I prefissi (Kilo, Mega, ecc.) sono normalmente associati a potenze di 10 mentre per i multipli del bit si opera su potenze di 2.

ATTENZIONE 1MB non corrisponde a 1000KB ma a 1024KB

Numeri binari negativi (1)

Nell'evoluzione dell'aritmetica binaria sono state utilizzate quattro rappresentazioni per i numeri negativi (Grandezza e segno, Complemento a uno, Complemento a due, Eccesso 2^{m-1}). Le diverse soluzioni mirano a ottimizzare le operazioni da svolgere sui numeri binari.

La codifica ideale prevede:

- Una sola rappresentazione per lo 0
- Lo stesso insieme di valori positivi e negativi rappresentabili

Questa codifica non è ottenibile per numeri binari poiché prevede di rappresentare un numero dispari di valori mentre avendo k bit a disposizione si potranno sempre rappresentare 2^k valori.

Grandezza e segno: con questa rappresentazione il bit più a sinistra viene utilizzato per il segno: con 0 si indicano i valori positivi mentre con 1 quelli negativi. I bit rimanenti contengono il valore assoluto del numero.

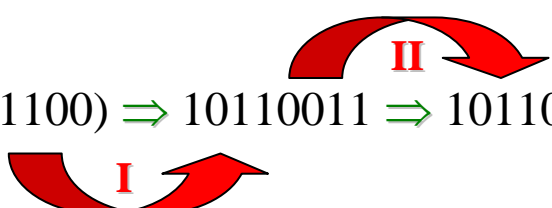
$$76 \Leftrightarrow 01001100$$

$$-76 \Leftrightarrow 11001100$$

Complemento a due: con questa rappresentazione il bit più a sinistra viene utilizzato per il segno: con 0 si indicano i valori positivi mentre con 1 quelli negativi. La negazione di un numero richiede due passi:

- I. Si sostituiscono tutti gli uno con degli zero e viceversa
- II. Si aggiunge 1 al risultato

Esempio: Rappresentazione in complemento a due di -76

$$-(76) \Rightarrow -(01001100) \Rightarrow 10110011 \Rightarrow 10110100$$


Numeri binari negativi (2)

Eccesso 2^{m-1} : rappresenta i numeri come somma di se stessi con 2^{m-1} dove m è il numero di bit utilizzati per rappresentare il valore. Si noti che il sistema è identico al complemento a due con il bit di segno invertito. In pratica i numeri tra -128 e 127 sono mappati tra 0 e 255 .

Esempio: Rappresentazione tramite eccesso a 2^{m-1} per -76 supponendo che il valore sia memorizzato in un byte ($m=8 \Rightarrow 2^{m-1}=2^7=128$)

$$-76 \Rightarrow -76+128 = 52 = 00110100$$

Decimale N	Binario N (senza rappresentare i numeri negativi)	Grand. e Segno -N	Complemento a due -N	Eccesso 128 -N
0	00000000	10000000	00000000	10000000
1	00000001	10000001	11111111	01111111
2	00000010	10000010	11111110	01111110
3	00000011	10000011	11111101	01111101
10	00001010	10001010	11110110	01110110
20	00010100	10010100	11101100	01101100
50	00110010	10110010	11001110	01001110
100	01100100	11100100	10011100	00011100
127	01111111	11111111	10000001	00000001
128	10000000	-	10000000	00000000

Si noti che:

- La rappresentazione grandezza e segno presenta due configurazioni diverse per lo zero: lo 0 positivo (00000000) e lo 0 negativo (10000000).
- Nelle rappresentazioni in complemento a due e in eccesso 2^{m-1} gli insiemi di valori positivi e negativi rappresentabili sono diversi poiché entrambe presentano una sola rappresentazione per lo 0.

Numeri binari negativi (3)

Le rappresentazioni in complemento a due ed eccesso 2^{m-1} sono le più efficienti per svolgere operazioni in aritmetica binaria poiché permettono di trattare la sottrazione tra numeri come una somma tra numeri di segno opposto.

$$(X - Y) = (X + (-Y))$$

Qualunque sia la rappresentazione utilizzata il numero di configurazioni rappresentabili non cambia ma, rispetto al caso in cui vengano rappresentati solo numeri positivi, l'intervallo positivo è dimezzato a favore dei valori negativi.

<i>n</i>	Configurazioni	Positivi	Positivi e Negativi
8	256	0 – 255	-128 – 127
16	65.536	0 – 65.535	-32768 – 32767
32	4.294.967.296	0 – 4.294.967.295	-2.147.483.648 – 2.147.483.647

Somma tra numeri binari

La tavola di addizione per i numeri binari è indicata di seguito:

	Configurazioni			
Addendo	0	0	1	1
Addendo	0	1	0	1
Somma	0	1	1	0
Riporto	0	0	0	1

Il procedimento di somma binaria è equivalente a quello nel sistema decimale con eccezione del riporto che si genera quando entrambi gli addendi hanno valore 1.

Operando con numeri in complemento a due:

- Il **riporto** generato dai bit più a sinistra viene ignorato.
- Se gli addendi sono di segno opposto non si può verificare un overflow.
- L'**overflow** si verifica se il riporto **generato** nel sommare i bit di segno è diverso dal riporto **utilizzato** per sommare i bit di segno (normalmente l'overflow viene indicato da un particolare bit di overflow del sommatore).

Addendo	54	00110110
Addendo	30	00011110
Somma	84	01010100
Riporto		00111110

Addendo	85	01010101
Addendo	74	01001010
Somma	159	10011111
Riporto		01000000

Addendo	10	00001010
Addendo	-3	11111101
Somma	7	00000111
Riporto		1111000

Overflow

Ignorato

Maschere dei bit (1)

Un byte (o una parola) fornisce un insieme di configurazioni che normalmente vengono utilizzate per codificare dei numeri. Tuttavia queste configurazioni possono essere utilizzate per memorizzare qualsiasi tipo di informazione.

00000001 Semaforo rosso 00000010 Semaforo giallo 00000100 Semaforo verde

In altri termini è sempre possibile associare ad alcuni numeri un particolare significato (Semaforo rosso=1, Semaforo giallo=2, Semaforo verde=4)

Chiameremo **maschera di bit** una qualsiasi configurazione di un insieme di bit di lunghezza predefinita.

Le operazioni che può essere utile eseguire su una locazione di memoria (byte o parola) utilizzata come maschera di bit sono:

- Set di uno o più bit
- Verifica del valore di uno o più bit

Queste operazioni possono essere eseguite tramite operatori che lavorano bit a bit (**bitwise operator**)

- **AND bit a bit:** dati due byte (parole) in input restituisce un byte (parola) in cui un bit è 1 se e solo se i bit corrispondenti nei due operandi sono posti a 1

00110110 AND 01111000 = 00110000

- **OR bit a bit:** dati due byte (parole) in input restituisce un byte (parola) in cui un bit è 1 se almeno uno dei bit corrispondenti nei due operandi sono posti a 1

00110110 OR 01111000 = 01111110

Maschere dei bit (2)

Impostare a 1 uno o più bit di una locazione di memoria X:

$X = X \text{ OR bit a bit}$ maschera in cui i bit di interesse sono impostati a 1 e i rimanenti a 0.

Esempio: Dato $X = 11001010$ impostare a 1 il terzo e quarto bit

Operando	11001010 Bitwise OR
Maschera	00001100
<hr/>	
Risultato	11001110

Impostare a 0 uno o più bit di una locazione di memoria X

$X = X \text{ AND bit a bit}$ maschera in cui i bit di interesse sono impostati a 0 e i rimanenti a 1.

Esempio: Dato $X = 00110101$ impostare a 0 il primo e l'ultimo bit

Operando	00110101 Bitwise AND
Maschera	01111110
<hr/>	
Risultato	00110100

Verificare se uno o più bit una locazione di memoria X sono impostati a 1

$Y = X \text{ AND bit a bit}$ maschera in cui gli unici bit impostati a 1 sono quelli di interesse. Se $Y=0$ la condizione non è verificata.

Verificare se uno o più bit una locazione di memoria X sono impostati a 0

$Y = X \text{ AND bit a bit}$ maschera in cui gli unici bit impostati a 1 sono quelli di interesse. Se $Y=0$ la condizione è verificata.

Esercizi

5. Quanti valori sono codificabili con 2-4-7-10 bit ?
6. Per codificare 1598 numeri quanti bit/byte sono necessari ?
7. Quante cifre binarie sono necessarie per codificare X numeri distinti ($\log_2 X = \log_{10} X / \log_{10} 2$)?
8. Si convertano i seguenti numeri decimali in numeri binari in complemento a 2: **-56 -88 -243**. Si utilizzino 8 bit per rappresentare il risultato.
9. Si convertano i seguenti numeri decimali in numeri binari in eccesso 128 ed eccesso 256 **-56 -37 -243**
10. Si pongano a 1 i bit di posizione dispari del byte **11010011**
11. Si pongano a 0 i primi 4 bit del byte **11010011**
12. Si verifichi che il terzo bit del byte **11001100** è 1
13. Si eseguano le seguenti somme in base 2:
 $12_{10} + 7_{10}$ $67_{10} + 38_{10}$ $89_{10} + 147_{10}$
14. Si eseguano le seguenti differenze in base 2 (complemento a 2, utilizzando 8 bit per rappresentare i numeri):
 $8_{10} - 17_{10}$ $37_{10} - 23_{10}$ $5_{10} - 117_{10}$

Numeri floating-point (1)

Molti problemi di calcolo richiedono una gamma di valori molto vasta difficilmente esprimibile con le convenzionali tecniche di numerazione:

- la massa del sole 2×10^{33} gr. richiede un numero a 34 cifre decimali oppure 111 cifre binarie
- la massa dell'elettrone 9×10^{-28} gr. richiede un numero con 28 decimali oppure 90 cifre binarie

Ovviamente manipolare questi valori *per esteso* è molto scomodo e spesso infattibile. Inoltre per numeri di queste dimensioni può essere inutile rappresentare le cifre meno significative (es. la massa del sole non è certa dopo le prime cinque cifre significative).

Si adotta quindi una notazione in cui la **gamma** dei valori esprimibili è indipendente dal numero di **cifre significative**. Questo sistema è detto **floating-point**

$$n = \underset{\substack{\uparrow \\ \text{frazione o mantissa}}}{f} \times 10^{\overset{\substack{\leftarrow \\ \text{esponente}}}{e}}$$

La precisione è determinata dalla mantissa f mentre la gamma dei valori è determinato dall'esponente e .

Esempio: Valori floating point corrispondenti alla mantissa $f=0.3682$ al variare del numero di cifre significative mantenute e al valore dell'esponente

$f \cdot e$	-4	-3	-2	-1	0	1	2	3	4
1	0.00003	0.0003	0.003	0.03	0.3	3	30	300	3000
2	0.000036	0.00036	0.0036	0.036	0.36	3.6	36	360	3600
3	0.0000368	0.000368	0.00368	0.0368	0.368	3.68	36.8	368	3680
4	0.00003682	0.0003682	0.003682	0.03682	0.3682	3.682	36.82	368.2	3682

Numeri floating-point (2)

La notazione floating point consente più rappresentazioni per uno stesso numero:

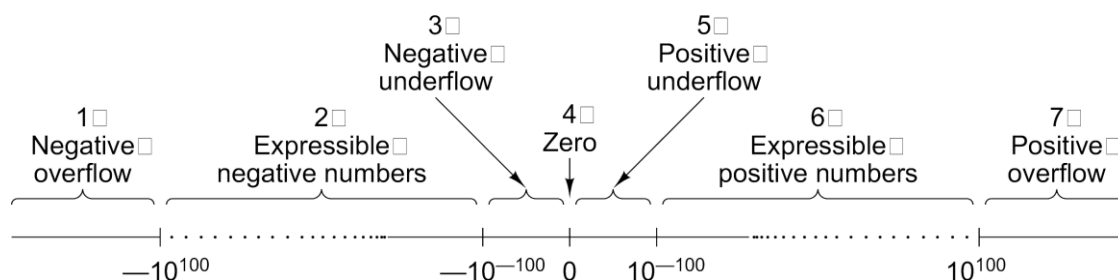
$$\begin{aligned} 3.14 &= 0.314 \times 10^1 = 3.14 \times 10^0 \\ 0.00001 &= 0.1 \times 10^{-4} = 1.0 \times 10^{-5} \\ 2987 &= 0.2987 \times 10^4 = 2.987 \times 10^3 \end{aligned}$$

Per questo motivo è stata definita una **codifica standard (o normalizzata)** in cui i valori della mantissa f sono compresi tra 0.1 e 1 ($0.1 \leq f < 1$).

I numeri floating-point possono essere utilizzati per “simulare” i numeri reali. Tuttavia la limitatezza delle cifre a disposizione inciderà sulla gamma dei valori rappresentabili o sulla loro precisione. A parità di cifre disponibili per memorizzare un numero floating-point, aumentando il numero di cifre impiegate per codificare la mantissa si ridurrà la gamma di valori rappresentabili e viceversa.

Si considerino i numeri floating-point rappresentabili con 3 cifre con segno nella mantissa e un esponente di 2 cifre con segno. L'insieme dei numeri reali risulta così suddiviso:

- | | |
|--|------------------------|
| (1) $r < -0.999 \times 10^{99}$ | Overflow |
| (2) $-0.999 \times 10^{99} \leq r \leq -0.1 \times 10^{-99}$ | Rappresentabile |
| (3) $-0.1 \times 10^{-99} < r < 0$ | Underflow |
| (4) $r = 0$ | Rappresentabile |
| (5) $0 < r < 0.1 \times 10^{-99}$ | Underflow |
| (6) $0.1 \times 10^{-99} \leq r \leq 0.999 \times 10^{99}$ | Rappresentabile |
| (7) $0.999 \times 10^{99} < r$ | Overflow |



Numeri floating-point (3)

Fissate le caratteristiche di un numero floating-point (numero di cifre della mantissa e dell'esponente) sono determinati anche gli intervalli di numeri reali rappresentabili. Oltre a ciò:

Non tutti i numeri reali appartenenti alle aree rappresentabili possono essere espressi correttamente tramite un numero floating-point

Esempio: Con numeri floating point con tre cifre decimali con segno per la mantissa e due cifre decimali con segno per l'esponente non è possibile rappresentare $10/3 = 3.\bar{3}$

$$0.333 \times 10^1 < 3.\bar{3} < 0.334 \times 10^1$$

A differenza dei numeri reali la **densità** dei numeri floating-point non è infinita.

Quando il risultato v non si può esprimere nella rappresentazione numerica che viene usata viene utilizzato il numero più vicino rappresentabile ($v_1 < v < v_2$). L'errore che si commette viene detto **errore di arrotondamento**.

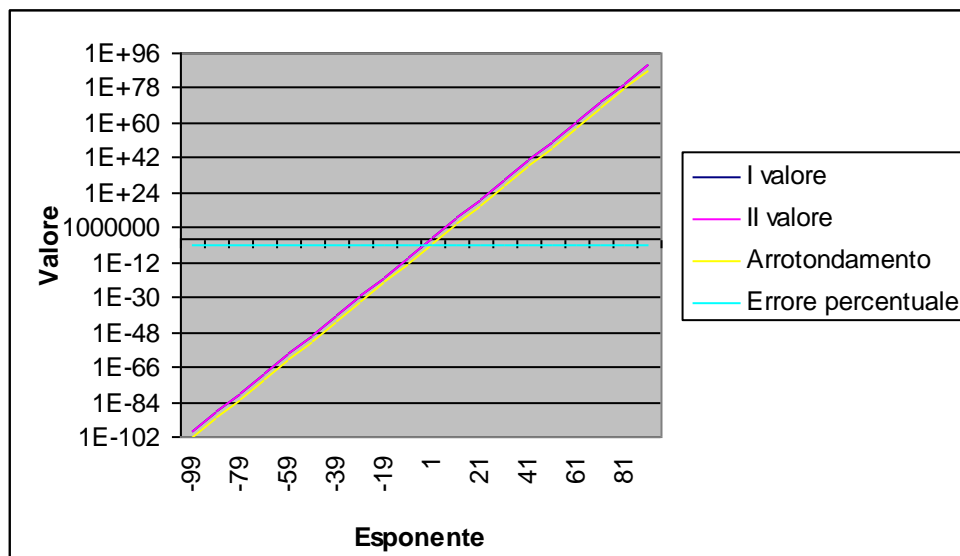
$$E = \min(|v - v_1|; |v - v_2|)$$

Esempio: Nel caso precedente l'errore di arrotondamento è

$$3.\bar{3} - 0.333 \times 10^1 = 0.000\bar{3}$$

Numeri floating-point (4)

Lo spazio tra numeri floating-point esprimibili non è costante e cresce al crescere dei valori rappresentati. Ciò che rimane costante è l'errore relativo, ossia l'errore di arrotondamento espresso in funzione del valore rappresentato.



Per incrementare la densità dei numeri è necessario aumentare il numero di cifre contenute nella mantissa.

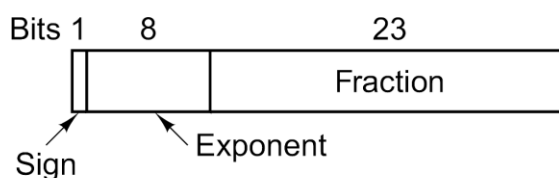
Fino a ora abbiamo operato con numeri floating-point decimali. I calcolatori operano con numeri floating point binari, in cui cioè mantissa ed esponente sono composti da cifre binarie.

IEEE 754: standard floating-point (1)

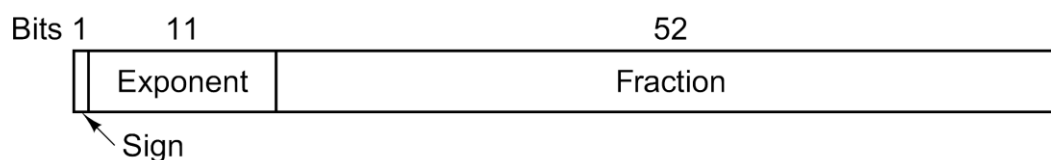
Fino all'introduzione dello standard IEEE (negli anni '80) ogni produttore di calcolatori aveva un proprio formato per la rappresentazione dei numeri floating-point binari.

Lo standard, studiato dal Prof. William Kahan, prevede tre formati: **singola precisione** (a), **doppia precisione** (b), **precisione estesa**. Le caratteristiche dei primi due formati sono presentate di seguito:

Elemento	Singola Precisione	Doppia Precisione
Num. bit nel segno	1	1
Num. bit nell'esponente	8	11
Num. Bit nella mantissa	23	52
Num. Bit totale	32	64
Rappresentazione dell'esponente	Eccesso 127	Eccesso 1023
Campo dell'esponente	Da -126 a +127	Da -1022 a +1023
Numero più piccolo	$2^{-126} \approx 10^{-38}$	$2^{-1022} \approx 10^{-308}$
Numero più grande	$\approx 2^{128} \approx 10^{38}$	$\approx 2^{1024} \approx 10^{308}$



(a)



(b)

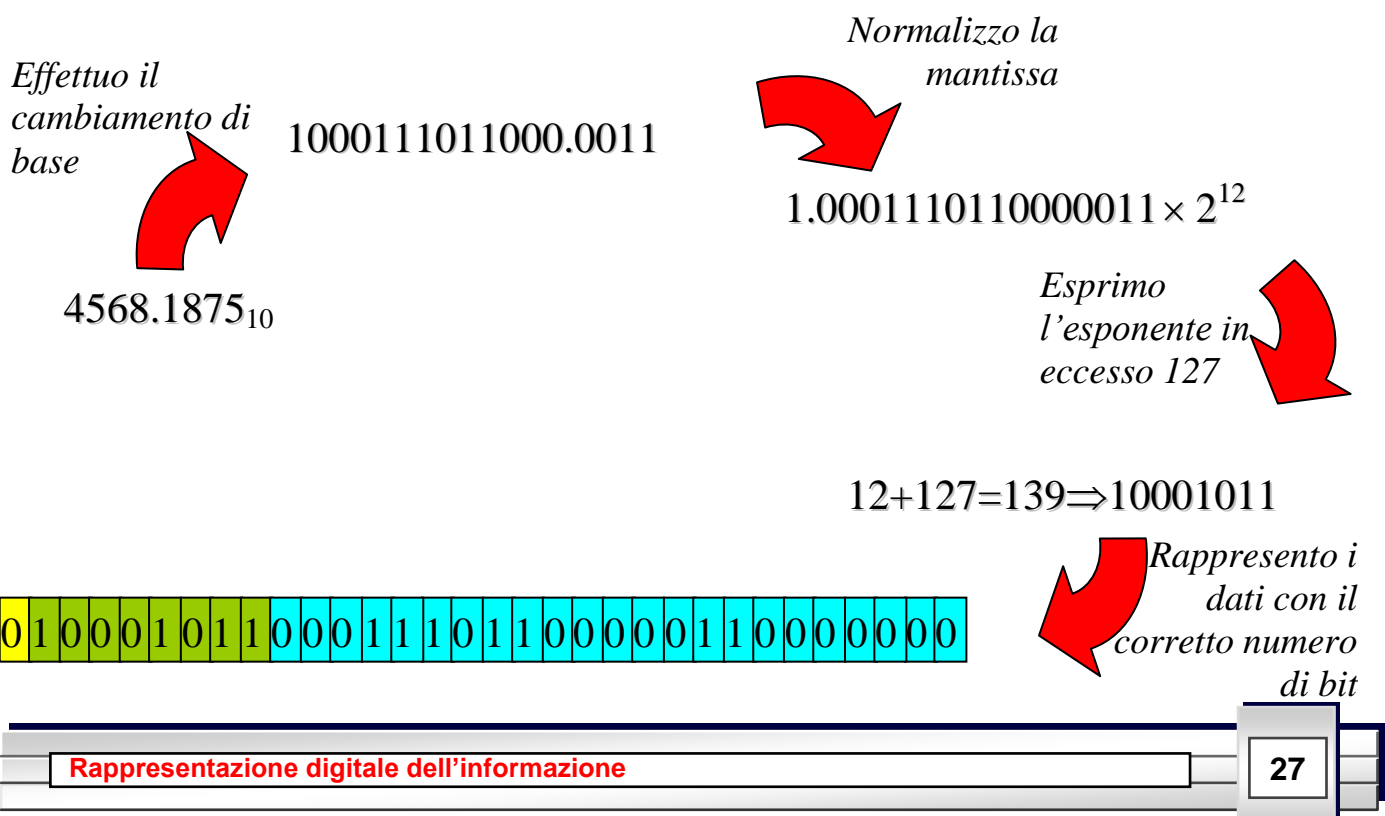
IEEE 754: standard floating-point (2)

Lo standard prevede di rappresentare i numeri in forma normalizzata e non normalizzata:

Forma Normalizzata: la mantissa binaria normalizzata deve presentare un 1 a sinistra della virgola binaria. L'esponente deve essere aggiustato di conseguenza.

- Essendo sempre presente tale cifra non è informativa così come la virgola binaria; esse vengono considerate implicitamente presenti e non vengono memorizzate.
- Per evitare confusione con una frazione tradizionale la combinazione dell'1 implicito della virgola binaria e delle 23/52 cifre significative vengono chiamate **significando** (invece che frazione o mantissa).
- Tutti i numeri normalizzati hanno un esponente $e > 0$
- Tutti i numeri normalizzati hanno un significando s tra $1 \leq s < 2$
- Il numero zero non è rappresentabile in questo modo: viene rappresentato con tutti i bit a zero sia nella mantissa che nell'esponente; a seconda del valore del bit di segno, si ottiene uno "zero negativo" o uno "zero positivo".
- I numeri normalizzati non possono avere un esponente composto da soli 1. Tale configurazione serve per modellare il valore infinito (∞).

Esempio: Trasformazione del numero 4568.1875_{10} in formato IEEE 754 in singola precisione



IEEE 754: standard floating-point (3)

I vincoli imposti dalla forma normalizzata non permettono di sfruttare appieno la gamma di valori rappresentabili con 32 e 64 bit. Ad esempio, in singola precisione non sono rappresentabili i numeri più piccoli di

$$1.0 \times 2^{-126}$$

rilassando il vincolo legato alla rappresentazione della mantissa questo problema può essere risolto e si potranno rappresentare i numeri fino a 2^{-150}

Forma Denormalizzata: la mantissa binaria denormalizzata può assumere qualsiasi configurazione. Questa rappresentazione viene utilizzata per rappresentare valori inferiori a 2^{-126}

- Tutti i bit dell'esponente sono posti a 0 (questa configurazione indica l'utilizzo della forma denormalizzata)
- Il bit della mantissa a sinistra della virgola binaria è posto implicitamente a 0
- Il numero più piccolo rappresentabile in questa configurazione è composto da una mantissa con tutti 0 a eccezione del bit più a destra.
- La rappresentazione denormalizzata comporta una progressiva perdita di cifre significative.

Esercizi

15. Quali numeri decimali sono codificabili in formato floating point utilizzando 5 cifre con segno per la mantissa e 3 cifre con segno per l'esponente?
16. Si calcoli l'errore di arrotondamento per la rappresentazione floating point decimale (3 cifre con segno per la mantissa e 2 cifre con segno per l'esponente) dei numeri:

1598 58978922 -0.568282

17. Si trasformino in formato IEEE 754 in singola precisione i numeri:

1598 -0.56640625

18. Si calcoli il valore decimale corrispondente ai seguenti numeri in formato IEEE 754 in singola precisione:

0|10011001|1011101110000000000000
1|01101101|1011101110010000000000

Rappresentazione di caratteri (1)

- Ogni calcolatore dispone di un set di caratteri da utilizzare.
- All'interno del calcolatore i caratteri sono codificati sotto forma di numeri.
- La mappatura dei caratteri in numeri è detta **codice di caratteri**.
- Per poter comunicare è essenziale che i calcolatori utilizzino la stessa mappatura.

ASCII (American Standard Code for Information Interchange)

- Ogni carattere ASCII utilizza 7 bit, sono quindi possibili 128 configurazioni.
- I caratteri da 0 a 1F (31_{10}) sono caratteri di controllo e non vengono stampati.
- Molti dei caratteri di controllo servono alla trasmissione dei dati nelle telescriventi ma, con l'evoluzione dei protocolli di comunicazione, non vengono più utilizzati.
- Dato che i dati vengono trasmessi in byte l'ottavo bit viene utilizzato per verificare la correttezza del dato inviato (controllo di parità).
- Dato che il formato ASCII non viene più utilizzato come protocollo di trasmissione, l'ottavo bit viene normalmente utilizzato per codificare caratteri non standard.
- Anche utilizzando 256 configurazioni non è possibile modellare tutti i simboli utilizzati nel mondo per comunicare (≈ 200.000)
- La prima estensione all'ASCII è stato il **Latin-1** che utilizza un codice a 8 bit per modellare anche lettere latine con accenti e segni diacritici (segni supplementari per precisare particolarità della pronuncia)
- In seguito fu ideato lo standard IS 8859 che tramite il concetto di **pagina di codice** assegnava un insieme di 256 caratteri a una lingua o insieme di lingue (Latin-1 è una pagina di codice)

Tabella caratteri ASCII standard

Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char
00000000	0	Null	00100000	32	Spa	01000000	64	@	01100000	96	`
00000001	1	Start of heading	00100001	33	!	01000001	65	A	01100001	97	a
00000010	2	Start of text	00100010	34	"	01000010	66	B	01100010	98	b
00000011	3	End of text	00100011	35	#	01000011	67	C	01100011	99	c
00000100	4	End of transmit	00100100	36	\$	01000100	68	D	01100100	100	d
00000101	5	Enquiry	00100101	37	%	01000101	69	E	01100101	101	e
00000110	6	Acknowledge	00100110	38	&	01000110	70	F	01100110	102	f
00000111	7	Audible bell	00100111	39	'	01000111	71	G	01100111	103	g
00001000	8	Backspace	00101000	40	(01001000	72	H	01101000	104	h
00001001	9	Horizontal tab	00101001	41)	01001001	73	I	01101001	105	i
00001010	10	Line feed	00101010	42	*	01001010	74	J	01101010	106	j
00001011	11	Vertical tab	00101011	43	+	01001011	75	K	01101011	107	k
00001100	12	Form Feed	00101100	44	,	01001100	76	L	01101100	108	l
00001101	13	Carriage return	00101101	45	-	01001101	77	M	01101101	109	m
00001110	14	Shift out	00101110	46	.	01001110	78	N	01101110	110	n
00001111	15	Shift in	00101111	47	/	01001111	79	O	01101111	111	o
00010000	16	Data link escape	00110000	48	0	01010000	80	P	01110000	112	p
00010001	17	Device control 1	00110001	49	1	01010001	81	Q	01110001	113	q
00010010	18	Device control 2	00110010	50	2	01010010	82	R	01110010	114	r
00010011	19	Device control 3	00110011	51	3	01010011	83	S	01110011	115	s
00010100	20	Device control 4	00110100	52	4	01010100	84	T	01110100	116	t
00010101	21	Neg. acknowledge	00110101	53	5	01010101	85	U	01110101	117	u
00010110	22	Synchronous idle	00110110	54	6	01010110	86	V	01110110	118	v
00010111	23	End trans. block	00110111	55	7	01010111	87	W	01110111	119	w
00011000	24	Cancel	00111000	56	8	01011000	88	X	01111000	120	x
00011001	25	End of medium	00111001	57	9	01011001	89	Y	01111001	121	y
00011010	26	Substitution	00111010	58	:	01011010	90	Z	01111010	122	z
00011011	27	Escape	00111011	59	;	01011011	91	[01111011	123	{
00011100	28	File separator	00111100	60	<	01011100	92	\	01111100	124	
00011101	29	Group separator	00111101	61	=	01011101	93]	01111101	125	}
00011110	30	Record Separator	00111110	62	>	01011110	94	^	01111110	126	~
00011111	31	Unit separator	00111111	63	?	01011111	95	_	01111111	127	Del

- Se si sta usando Windows si può ottenere ogni carattere ASCII tenendo premuto il tasto **Alt** e digitando il codice decimale corrispondente con il tastierino numerico (@=**Alt 64**).
- Nella tastiera inglese sono già presenti tutti i caratteri della tabella standard; nella tastiera italiana invece mancano l'apice (96), le parentesi graffe (123,125) e la tilde (126).

Tabella caratteri ASCII estesa

Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char
10000000	128	Ç	10100000	160	á	11000000	192	+	11100000	224	Ó
10000001	129	ü	10100001	161	í	11000001	193	-	11100001	225	ß
10000010	130	é	10100010	162	ó	11000010	194	-	11100010	226	Ô
10000011	131	â	10100011	163	ú	11000011	195	+	11100011	227	Ò
10000100	132	ä	10100100	164	ñ	11000100	196	-	11100100	228	ö
10000101	133	à	10100101	165	Ñ	11000101	197	+	11100101	229	Õ
10000110	134	å	10100110	166	ª	11000110	198	ä	11100110	230	µ
10000111	135	ç	10100111	167	•	11000111	199	Ä	11100111	231	þ
10001000	136	ê	10101000	168	¿	11001000	200	+	11101000	232	ð
10001001	137	ë	10101001	169	@	11001001	201	+	11101001	233	Ù
10001010	138	è	10101010	170	¬	11001010	202	-	11101010	234	Û
10001011	139	ï	10101011	171	½	11001011	203	-	11101011	235	Ü
10001100	140	î	10101100	172	¾	11001100	204	!	11101100	236	ý
10001101	141	ì	10101101	173	¡	11001101	205	-	11101101	237	Ý
10001110	142	Ä	10101110	174	«	11001110	206	+	11101110	238	—
10001111	143	Å	10101111	175	»	11001111	207	©	11101111	239	·
10010000	144	É	10110000	176	-	11010000	208	ð	11110000	240	-
10010001	145	æ	10110001	177	-	11010001	209	Ð	11110001	241	±
10010010	146	Æ	10110010	178	-	11010010	210	Ê	11110010	242	-
10010011	147	ô	10110011	179	-	11010011	211	Ë	11110011	243	¼
10010100	148	ö	10110100	180	-	11010100	212	È	11110100	244	¶
10010101	149	ò	10110101	181	Á	11010101	213	í	11110101	245	§
10010110	150	û	10110110	182	Â	11010110	214	Î	11110110	246	÷
10010111	151	ù	10110111	183	Ã	11010111	215	Ï	11110111	247	,
10011000	152	ÿ	10111000	184	©	11011000	216	Ï	11111000	248	ó
10011001	153	Ö	10111001	185	!	11011001	217	+	11111001	249	"
10011010	154	Ü	10111010	186	!	11011010	218	+	11111010	250	.
10011011	155	ß	10111011	187	+	11011011	219	-	11111011	251	1
10011100	156	£	10111100	188	+	11011100	220	-	11111100	252	3
10011101	157	Ø	10111101	189	¢	11011101	221	-	11111101	253	2
10011110	158	×	10111110	190	¥	11011110	222	Ì	11111110	254	-
10011111	159	f	10111111	191	+	11011111	223	-	11111111	255	-

Rappresentazione di caratteri (2)

UNICODE

- Sistema per la codifica dei caratteri di quasi tutte le lingue vive (e alcune lingue morte), includendo anche simboli matematici, alfabeto Braille, ideogrammi, simboli cartografici, etc.
- In origine (v1.0 del 1991) era stato pensato come una codifica a 16 bit, quindi con 65.536 simboli (chiamati **code point**), che all'epoca si riteneva sufficiente.
- Dalla v2.0 (1996) fu prevista la possibilità di avere code point oltre il range 0000-FFFF, quindi codificati con un numero di bit maggiore di 16.
- Nella v10.0 (2017) sono presenti più di 136.000 code point, con codifica fino a 21 bit.
- Si tratta di uno standard internazionale (viene mantenuto allineato con lo standard **ISO/IEC 10646**).
- Varie codifiche possibili:
 - **UTF-8** (unità da 8 bit, 1/2/3/4 byte per code point): utilizzato per compatibilità (i primi 128 code point corrispondono al codice ASCII); tipicamente adottato nei file txt/html/xml.
 - **UTF-16** (unità da 16 bit, 2/4 byte per code point): utilizzato dai moderni sistemi operativi e linguaggi di programmazione per codifica di stringhe
 - **UTF-32** (unità da 32 bit): tutti i code point sono codificati in 4 byte.

Code range (hexadecimal)	UTF-8	UTF-16	UTF-32
000000 – 00007F	1	2	4
000080 – 00009F	2		
0000A0 – 0003FF			
000400 – 0007FF			
000800 – 003FFF	3		
004000 – 00FFFF			
010000 – 03FFFF	4	4	
040000 – 10FFFF			

UNICODE: Esempi

- Alcuni esempi di caratteri e codifiche:

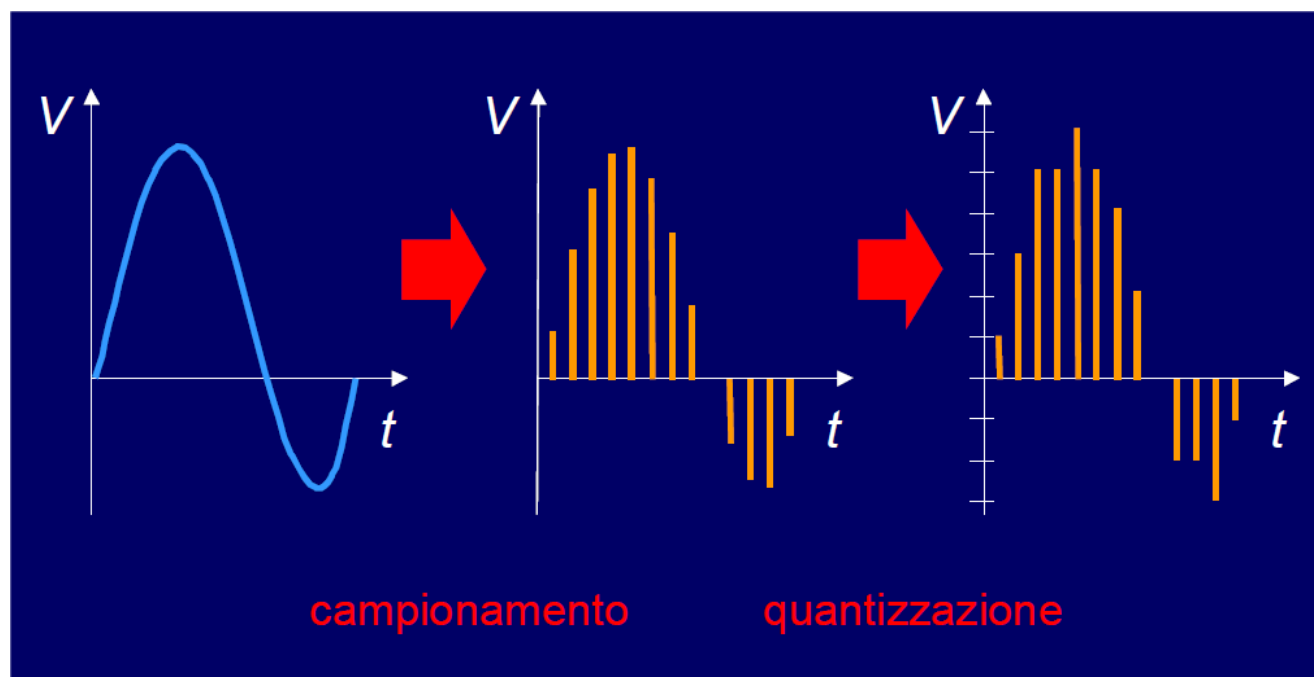
Character	Code Point	UTF-16	UTF-8
a	U+0061	0061	61
ä	U+00E4	00E4	C3 A0
σ	U+03C3	03C3	CF 83
κ	U+05D0	05D0	D7 90
۳	U+0663	0663	D9 A3
力	U+30AB	30AB	E3 82 AB
退	U+9000	9000	E9 80 80
ㄅ	U+21BC1	D846 DFC1	F0 A1 AF 81

- Nota bene - “code point” e “carattere” non sono sinonimi:
 - Alcuni “caratteri” sono ottenibili combinando più code points, ad es. Ĵ è ottenuto da due code points: (U+004A (J) seguito da U+030C (segno diacritico combinabile ˇ));
 - Alcuni code point corrispondono a simboli che l’utente percepisce come più caratteri affiancati, ad es. >>> è un singolo code point (U+22D9).

Rappresentazione di Suoni

Fisicamente un suono è rappresentato come un'onda (onda sonora) che descrive la variazione della pressione dell'aria nel tempo. Si tratta di un segnale analogico.

Codifica digitale



Parametri importanti della codifica (digitale):

- Frequenza di campionamento
- Numero di bit per la quantizzazione di ciascun campione

Il campionamento ad elevata frequenza e la quantizzazione con numero elevato di bit per campione garantiscono rappresentazione accurata.

Campionamento Audio qualità CD:

- 44,1 KHz (44.100 campioni al secondo) circa il doppio dei 22 KHz che rappresentano la banda udibile dall'uomo.
- 16 bit per canale (Stereo = 2 Canali)
- Ogni secondo di audio $\rightarrow 44100 \times (16/8) \times 2 \approx 172$ KB/sec

Rappresentazione di Immagini

L'immagine del video è rappresentata tramite una **griglia** o **matrice di pixel** (Picture Element) di ognuno dei quali è memorizzata l'intensità luminosa (e/o il colore).

Parametri importanti sono:

- Dimensione (risoluzione)
- Profondità
- Formato di rappresentazione (Grayscale, RGB, Palette)

Immagini **a livelli di grigio (Grayscale)**:

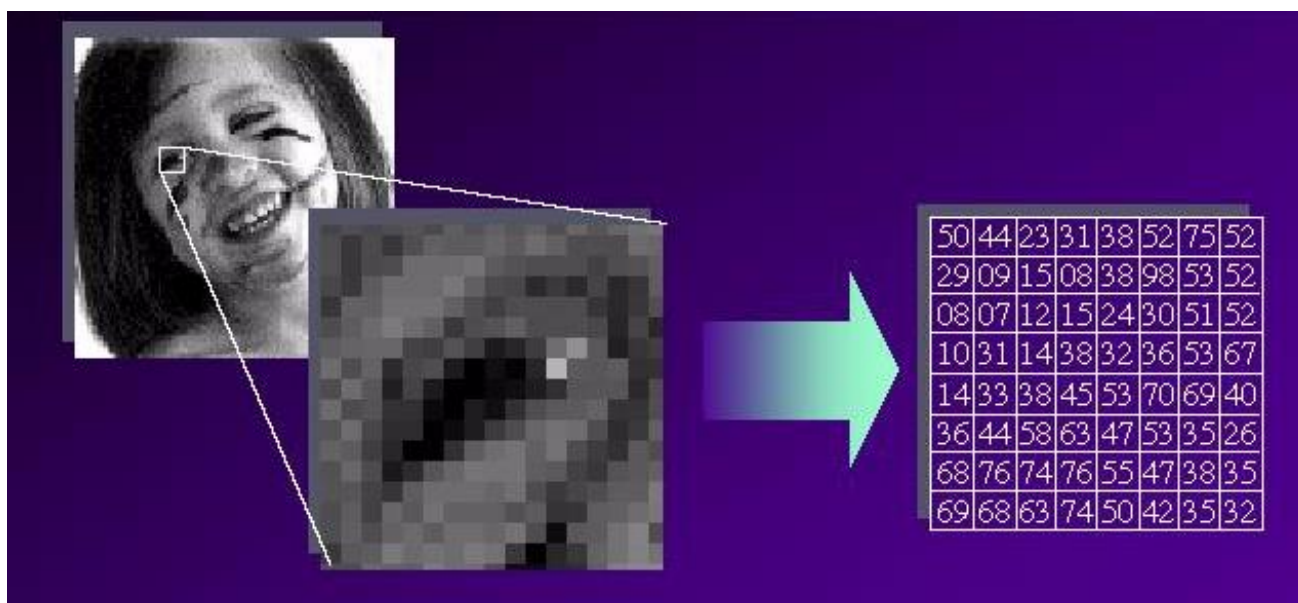
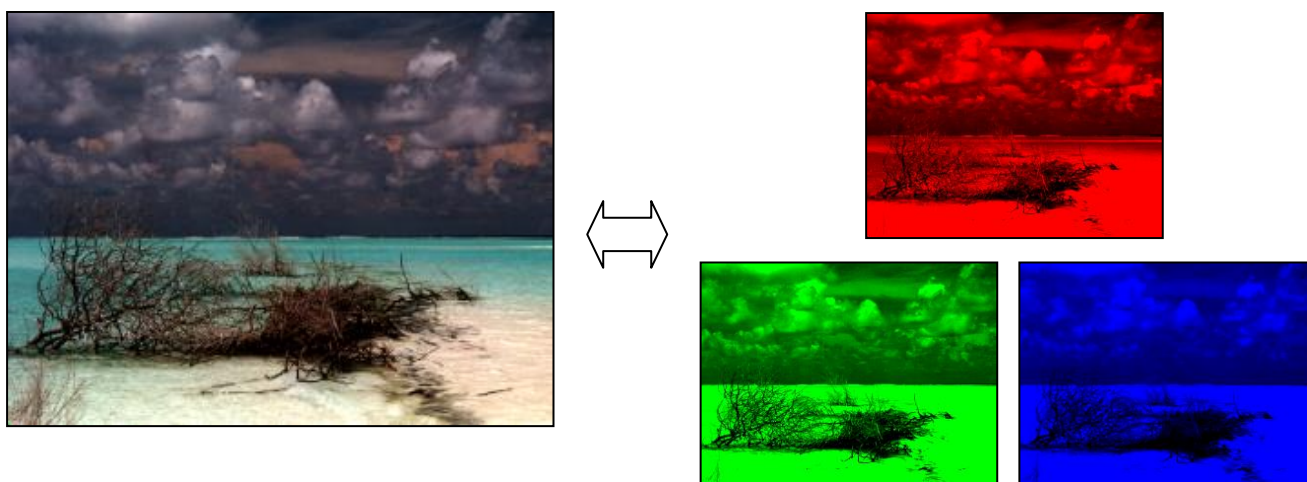


Immagine **a colori (RGB)**:



Immagini: codifica

Codifica immagini grayscale

- 1 pixel codificato con **un byte** → **256** livelli di grigio
- Livello 0 → Nero
- Livello 255 → Bianco
- Un'immagine di dimensioni $W \times H$ occupa **$W \times H$** bytes.

Codifica immagini RGB (24)

- 1 pixel codificato con **tre byte (R,G,B)** ciascuno dei quali specifica l'intensità di uno dei tre colori fondamentali (sintesi additiva)
- Il numero di colori disponibili è $2^{24} =$ **16.777.216**
- Livello 0,0,0 → Nero
- Livello 255,255,255 → Bianco
- X,X,X → Grigio
- Un'immagine di dimensioni $W \times H$ occupa **$W \times H \times 3$** byte.

Codifica immagini con un numero arbitrario di colori

- Supponiamo di utilizzare N bit per codificare ciascun pixel → il numero di colori disponibili è 2^N
- Un'immagine di dimensioni $W \times H$ occupa **$(W \times H \times N) / 8$** bytes;

Immagini: codifica (2)

Codifica tramite Palette

Per ridurre la quantità di memoria richiesta viene spesso utilizzata una **palette di colori**.

Una **palette** è una tabella che definisce 2^c colori (ognuno caratterizzato da una tripletta RGB, con profondità p bit) e associa a ciascun colore un **indice** (posizione nella palette). In un'immagine "palettizzata" il colore di **ogni pixel è definito dall'indice del colore nella palette** (*sono necessari c bit invece di p bit per ogni pixel!*). In questo modo in un'immagine, pur potendo utilizzare un numero massimo (2^c) di colori diversi, questi potranno essere scelti da un insieme molto più ampio (2^p).

Se $p = 24$ e $c = 8$ allora,
l'immagine potrà utilizzare 256 colori diversi scelti tra 16.777.216