



**UNIMORE**  
UNIVERSITÀ DEGLI STUDI DI  
MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche,  
Informatiche e Matematiche

## 4. Rappresentazione dell'informazione

### Architettura dei calcolatori [MN1-1143]

*Corso di Laurea in INFORMATICA*  
(D.M.270/04) [16-215]  
Anno accademico 2020/2021

**Prof. Andrea Marongiu**  
[andrea.marongiu@unimore.it](mailto:andrea.marongiu@unimore.it)

*È vietata la copia e la riproduzione dei contenuti e immagini in qualsiasi forma.*

*È inoltre vietata la redistribuzione e la pubblicazione dei contenuti e immagini non autorizzata espressamente dall'autore o dall'Università di Modena e Reggio Emilia.*

# Rappresentazione binaria

- Tutta l'informazione interna ad un computer è codificata con sequenze di due soli simboli: 0 e 1
  - è facile realizzare dispositivi elettronici che discriminano fra due stati, molto meno se gli stati sono tanti
- L'unità elementare di informazione si chiama **bit**
  - da '*binary digit*'
- **Byte**: sequenza di 8 bit
- **word**: sequenza di 32, 64, ... bits (4, 8, ... **Bytes**)
  - È la *parola* con cui un calcolatore sa lavorare
    - leggere/scrivere dalla memoria, elaborare...

# Rappresentazione binaria

- **word**: sequenza di 32, 64, ... bits (4, 8, ... **Bytes**)
  - È la *parola* con cui un calcolatore sa lavorare
    - **Processori a 8bit** (parole da 1Byte): Intel8080, Zilog Z80



1975

1994

- **Processori a 16bit** (parole da 2Byte): Intel 8086, Motorola 68000

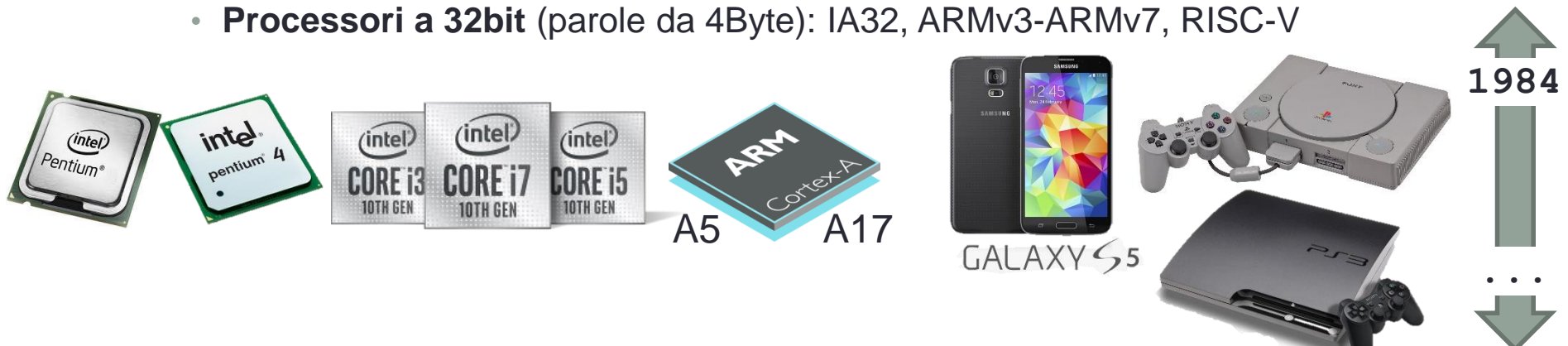


1979

1994

# Rappresentazione binaria

- **word**: sequenza di 32, 64, ... bits (4, 8, ... **Bytes**)
  - È la *parola* con cui un calcolatore sa lavorare
    - **Processori a 32bit** (parole da 4Byte): IA32, ARMv3-ARMv7, RISC-V



- **Processori a 64bit** (parole da 8Byte): AMD64, IA64, SPARCV9, PowerPC



# Sistema decimale posizionale (1)

- Un numero (es. 5) può essere rappresentato in molti modi:



- cinque, five, 5, V, XXXXX .....
- Rappresentazioni diverse hanno proprietà diverse
  - moltiplicare due numeri in notazione romana è molto più difficile che moltiplicare due numeri in notazione decimale ....
- Noi siamo abituati a lavorare con numeri rappresentati in notazione posizionale in base 10

# Sistema decimale posizionale (2)

- La rappresentazione di un numero intero in base 10 è una sequenza di cifre scelte fra l'insieme **{0 1 2 3 4 5 6 7 8 9}**
- Il valore di una rappresentazione

$$a_N a_{N-1} \dots a_0 \quad , \quad a_{-1} a_{-2} a_{-3} a_{-4} \dots$$

è dato da

$$\begin{aligned} & a_N \cdot 10^N + a_{N-1} \cdot 10^{N-1} + \dots + \text{(parte intera)} \\ & \quad + a_1 \cdot 10^1 + a_0 \cdot 10^0 \\ & \quad + \\ & \quad a_{-1} \cdot 10^{-1} + a_{-2} \cdot 10^{-2} + a_{-3} \cdot 10^{-3} + a_{-4} \cdot 10^{-4} + \dots \\ & \quad \text{(parte frazionaria)} \end{aligned}$$

- $b = 10$  è la **base**
- $10^i$  è il **peso** della cifra  $a_i$  nel valore del numero
- **Sistema decimale posizionale**

# Sistema decimale posizionale (3)

- **253** =  $2 \times 100 + 5 \times 10 + 3 \times 1 =$   
=  $2 \times 10^2 + 5 \times 10^1 + 3 \times 10^0$
- **23,47** =  $2 \times 10 + 3 \times 1 + 4 \times 0.1 + 7 \times 0.01 =$   
=  $2 \times 10 + 3 \times 1 + 4 \times (1/10) + 7 \times (1/100) =$   
=  $2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 7 \times 10^{-2}$
- Sistema **posizionale** (romani: sistema non posizionale)
- Sistema **decimale** (maya: sistema non decimale)



# Sistema decimale posizionale (4)

Alcune proprietà di questa notazione :

- Il massimo numero rappresentabile con  $N$  cifre è **99...9** ( $N$  volte 9, la cifra che vale di più), pari a  $10^N - 1$ 
  - **es:** su tre cifre il massimo numero rappresentabile è **999** pari a  $10^3 - 1 = 1000 - 1$
- Quindi se voglio rappresentare  $K$  diversi numeri (cioè  $0, 1, 2, \dots, K-1$ ) mi servono almeno almeno  $x$  cifre dove  $10^x$  è la più piccola potenza di 10 che supera  $K$ 
  - **es:** se voglio 25 configurazioni diverse mi servono almeno 2 cifre perché  $10^2 = 100$  è la più piccola potenza di 10 maggiore di 25

# Notazione posizionale in base 2 (1)

- La rappresentazione di un numero intero in **base 2** è una sequenza di cifre scelte fra **{0,1}** :
  - es: 10, 110, 1

- Il valore di una rappresentazione

$a_N \dots a_0, a_{-1} a_{-2} a_{-3} a_{-4}$

è dato da

$$\begin{aligned} & a_N \cdot 2^N + a_{N-1} \cdot 2^{N-1} + \dots + a_1 \cdot 2^1 + a_0 \cdot 2^0 + \\ & a_{-1} \cdot 2^{-1} + a_{-2} \cdot 2^{-2} + a_{-3} \cdot 2^{-3} + a_{-4} \cdot 2^{-4} + \dots \end{aligned}$$

(parte intera)

(parte frazionaria)

- b = 2** è la *base*, **2<sup>i</sup>** è il *peso* della cifra **a<sub>i</sub>** nel valore del numero

# Notazione posizionale in base 2 (2)

Esempi :

- 10  $= 1 \cdot 2^1 + 0 \cdot 2^0 = 2$
- 110  $= 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 4 + 2 + 0 = 6$
- 1  $= 1 \cdot 2^0 = 1$

*10 si legge “uno-zero” e non “dieci” !!!*

# Notazione posizionale in base 2 (3)

Per la base 2 valgono proprietà analoghe a quelle viste per la base 10:

- Il massimo numero rappresentabile con **N** cifre è **11...1** (N volte 1, la cifra che vale di più), pari a  **$2^N - 1$** 
  - **es:** *su tre cifre il massimo numero rappresentabile è **111** pari a  $2^3 - 1 = 8 - 1 = 7$*
- Quindi se voglio rappresentare **K** diversi numeri (cioè 0, 1, 2, ..., K-1) mi servono almeno almeno **x** cifre dove  **$2^x$**  è la più piccola potenza di 2 che supera **K**
  - **es :** *se voglio 25 configurazioni diverse mi servono almeno 5 cifre perché  $2^5=32$  è la più piccola potenza di 2 maggiore di 25*

# Conversione di interi: Base 10 $\rightarrow$ Base 2

- Successione di divisioni per 2:
  - termina quando il resto è 0
- La conversione in binario si ottiene leggendo i resti determinati in ordine inverso

**es.:**

13					Quozienti
					Resti

$$13_{10} = ( \quad )_2$$

# Conversione di interi: Base 10 $\rightarrow$ Base 2

- Successione di divisioni per 2:
  - termina quando il resto è 0
- La conversione in binario si ottiene leggendo i resti determinati in ordine inverso

**es.:**

					<b>/ 2</b>	
13	6				Quozienti	
					Resti	
	1					

$$13_{10} = ( \quad \mathbf{1} )_2$$

# Conversione di interi: Base 10 $\rightarrow$ Base 2

- Successione di divisioni per 2:
  - termina quando il resto è 0
- La conversione in binario si ottiene leggendo i resti determinati in ordine inverso

es.:

	<i>/ 2</i>	<i>/ 2</i>			
13	6	3			Quozienti
	<i>1</i>	<i>0</i>			
					Resti

$$13_{10} = ( \textcolor{red}{01} )_2$$

# Conversione di interi: Base 10 $\rightarrow$ Base 2

- Successione di divisioni per 2:
  - termina quando il resto è 0
- La conversione in binario si ottiene leggendo i resti determinati in ordine inverso

es.:

	<i>/ 2</i>	<i>/ 2</i>	<i>/ 2</i>		
13	6	3	1		Quozienti
	<i>1</i>	<i>0</i>	<i>1</i>		Resti

$$13_{10} = ( \textcolor{red}{101} )_2$$



# Conversione di interi: Base 10 $\rightarrow$ Base 2

- Successione di divisioni per 2:
  - termina quando il resto è 0
- La conversione in binario si ottiene leggendo i resti determinati in ordine inverso

es.:

	<i>/ 2</i>	<i>/ 2</i>	<i>/ 2</i>	<i>/ 2</i>	
13	6	3	1	0	Quozienti
	1	0	1	1	Resti

$$13_{10} = (1101)_2$$

# Conversione di interi: Base 2 $\rightarrow$ Base 10

- Somma pesata delle cifre binarie:

$$\begin{aligned}\text{es.: } 1101_2 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 8 + 4 + 0 + 1 \\ &= 13_{10}\end{aligned}$$

# Numeri binari interi: esempi

0	<b>0</b>	8	<b>1000</b>	16	<b>10000</b>
1	<b>1</b>	9	<b>1001</b>	17	<b>10001</b>
2	<b>10</b>	10	<b>1010</b>	18	<b>10010</b>
3	<b>11</b>	11	<b>1011</b>		... .
4	<b>100</b>	12	<b>1100</b>		
5	<b>101</b>	13	<b>1101</b>		
6	<b>110</b>	14	<b>1110</b>		
7	<b>111</b>	15	<b>1111</b>		

# Numeri binari interi: esempi(2)

$$\begin{array}{rcl} 2^0 & = & 1 \\ 2^1 & = & 2 \\ 2^2 & = & 4 \\ 2^3 & = & 8 \\ 2^4 & = & 16 \\ 2^5 & = & 32 \\ 2^6 & = & 64 \\ 2^7 & = & 128 \end{array}$$

$$\begin{array}{rcl} 2^8 & = & 256 \\ 2^9 & = & 512 \\ 2^{10} & = & 1024 \\ 2^{11} & = & 2048 \\ 2^{12} & = & 4096 \end{array}$$

...

$$2^{16} = 65536$$

...

$$2^{24} \cong 16 \text{ milioni}$$

...

# Aritmetica binaria

- Necessità di codificare nel mondo dei numeri binari ogni operazione aritmetica.

- **addizione:**

$$0+0=0$$

$$0+1=1$$

$$1+0=1$$

$$1+1=0 \text{ col riporto di } 1$$

- **sottrazione**

$$0-0=0$$

$$0-1=1 \text{ col prestito di } 1 \text{ dalla cifra precedente}$$

$$1-0=1$$

$$1-1=0$$

# Aritmetica binaria

- Necessità di codificare nel mondo dei numeri binari ogni operazione aritmetica.

- **addizione:**

$$0+0=0$$

$$0+1=1$$

$$1+0=1$$

$$1+1=0 \text{ col riporto di } 1$$

$$\begin{array}{r} 111 \\ 0101 + \\ 0011 = \\ \hline 1000 \end{array} \quad \begin{array}{r} 5_{10} + \\ 3_{10} = \\ \hline 8_{10} \end{array}$$

- **sottrazione**

$$0-0=0$$

$$0-1=1 \text{ col prestito di } 1 \text{ dalla cifra precedente}$$

$$1-0=1$$

$$1-1=0$$

# Aritmetica binaria (2)

- **moltiplicazione:**

- **es.:** per  $2, 2^2, 2^3, \dots \leftrightarrow$  **shift** (traslazione) verso sinistra di 1, 2, 3 bit

$$1101 \times 100 = 110100$$

$$(13 \times 4 = 52)$$

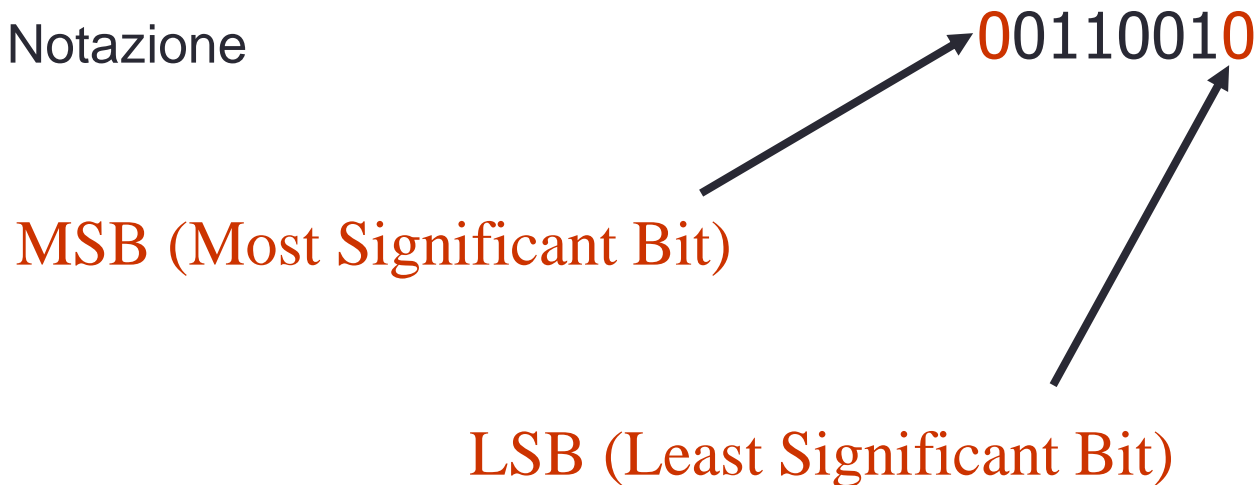
# La rappresentazione dei numeri all'interno di un computer

- Usa la notazione binaria
- Ogni numero viene rappresentato con un numero finito di cifre binarie (*bit*)
- Numeri di 'tipo' diverso hanno rappresentazioni diverse
  - **es.** interi positivi, interi (pos. e neg.), razionali, reali, complessi



# La rappresentazione dei numeri all'interno di un computer (1)

- Abbiamo già incontrato alcuni termini utili:
  - **byte** : una sequenza di 8 bit
  - **word** (parola) : 2, 4, 8 byte (dipende dalla macchina) unità minima che può essere fisicamente letta o scritta nella memoria (ed elaborata)
- Tipicamente gli interi positivi si rappresentano usando 4 o 8 byte
  - Esistono varianti a 2 byte (es. il tipo short int in C)
- Notazione



# La rappresentazione dei numeri all'interno di un computer (2)

- Alcuni punti importanti:
  - se uso 4 byte (32 bit) posso rappresentare solo i numeri positivi da 0 a  $2^{32}-1$ , che sono molti ma non *tutti* !
  - se moltiplico o sommo due numeri molto elevati posso ottenere un numero che non è rappresentabile
    - **es:** vediamo cosa succede in base 10 con solo 3 cifre :  
 $500 + 636 = 1136$  risultato 136

se uso solo 3 cifre non ho lo spazio fisico per scrivere la prima cifra (1) che viene 'persa', è un fenomeno chiamato **overflow**

$$\begin{array}{r} 101 + \\ 110 = \\ \hline 1011 \end{array}$$

il primo 1 non trova spazio

# La rappresentazione dei numeri all'interno di un computer (3)

- Interi positivi e negativi:
  - ci sono diverse convenzioni di rappresentazione
    - *modulo e segno* in cui il primo bit viene riservato al segno (1 negativo, 0 positivo) e gli altri 31 al modulo
    - *Complemento a due*
    - *Complemento a uno* (la trascuriamo)
  - rimane comunque il problema dell'overflow

# La rappresentazione dei numeri all'interno di un computer (4)

## Numeri relativi

- **Modulo e segno** (es con 3 bit)
  - 0 segno +
  - 1 segno -
- codifica semplice
- operazioni aritmetiche più complesse

es.:  $+2 \leftrightarrow 010$  e  $-2 \leftrightarrow 110$

$$\begin{array}{r} 001 + \\ 110 = \\ \hline 111 \end{array}$$

$$\begin{array}{r} 1 + \\ -2 = \\ \hline -3 \end{array}$$

Errato!

- Occorre differenziare tra i bit del numero e quelli di segno
- Bisogna codificare in modo diverso le operazioni aritmetiche.

# La rappresentazione dei numeri all'interno di un computer (5)

- **Complemento a due** (es con 4 bit)

es:  $+5 = 0101$

*Come si rappresenta -5 ??*

- Partendo da  $+5 = 0101$ :
  1. *si invertono gli 1 con gli 0: 1010*
  2. *si aggiunge 1:*

$$1010 + 1 = 1011 = -5$$

$$- 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

$$= - 8 + 0 + 2 + 1 = -5$$

- **Il primo bit non rappresenta solo il segno!**
- Non occorre più pertanto differenziare i bit.

# Esempio

$$\begin{array}{r} 0001 + \\ 1011 = \\ \hline 1100 \end{array}$$

$$\begin{array}{r} 1 + \\ -5 = \\ \hline -4 \end{array}$$

OK!

# Altre basi numeriche utilizzate

- **Ottale** (base 8): { 0, 1, 2, 3, 4, 5, 6, 7}     ( $10 \leftrightarrow 8$ )
- **Esadecimale** (base 16): { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}  
( $10 \leftrightarrow 16$ )
- Usate perché semplici conversioni da base 2 a base 8 o 16:
  - Partendo dal numero in base 2:
    1. *Se ne raggruppano le cifre a blocchi di 3 (ottale) o 4 (esadecimale)*
    2. *Si convertono i singoli gruppi nella base di destinazione*

# Altre basi numeriche utilizzate

Esempio:  $111000110101_2$

Conversione in base 8

$$\underbrace{111} \underbrace{000} \underbrace{110} \underbrace{101} = 7065_8$$

Conversione in base 16




$$\underbrace{1110} \underbrace{0011} \underbrace{0101} = E35_{16}$$



# Ulteriore sistema di codifica dei numeri

## **BCD** (Binary-Coded Decimal)

- Si codificano in binario (4 bit) le singole cifre decimali.
- **es.:** 254

		
2	5	4
0010	0101	0100

- nessun errore di conversione
- precisione dei calcoli decimali
- spreco di cifre
- usato nelle calcolatrici tascabili

# La rappresentazione dei numeri all'interno di un computer (6)

- **Razionali**

- numero finito di cifre periodiche dopo la virgola
  - *ad esempio* **3.12** oppure **3.453**
- rappresentazione solitamente su 4/8 byte

- **Rappresentazione in *virgola fissa* :**

- riservo X bit per la parte frazionaria
- *es : con 3 bit per la parte intera e 2 per quella frazionaria* **011.11, 101.01**



# La rappresentazione dei numeri all'interno di un computer (7)

- Come si converte in base 10 una rappresentazione in virgola fissa?

- **es:**

$$\begin{aligned} 101.01 &= 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} = \\ &= 4 + 0 + 1 + 0 + 0.25 = 5.25 \end{aligned}$$

dove  $2^{-1} = 1/2 = 0.5$ ,  $2^{-2} = 1/2^2 = 0.25$

e in generale  $2^{-n} = 1/2^n$

# La rappresentazione dei numeri all'interno di un computer (8)

- Problemi della rappresentazione in virgola fissa
  - **overflow**
  - **underflow**
  - quando si scende al di sotto del minimo numero rappresentabile
    - es. vediamo in base 10, con 2 cifre riservate alla parte frazionaria  
 $0.01 / 2 = 0.005$  **non rappresentabile usando solo due cifre!!!**

# La rappresentazione dei numeri all'interno di un computer (9)

- Problemi della rappresentazione in virgola fissa (cont.)
  - spreco di bit per memorizzare molti '0' quando lavoro con numeri molto piccoli o molto grandi
    - *es. in base 10, con 5 cifre per la parte intera e 2 cifre per la parte frazionaria*  
*10000.00 oppure 00000.02*
  - i bit vengono usati più efficientemente con la notazione *esponenziale* o ***floating point*** (*virgola mobile*)

# La rappresentazione dei numeri all'interno di un computer (10)

- **Rappresentazione in *virgola mobile***

- **idea:**

- quando lavoro con numeri molto piccoli uso tutti i bit disponibili per rappresentare le cifre dopo la virgola
    - quando lavoro con numeri molto grandi le uso tutte per rappresentare le cifre in posizioni elevate
  - questo permette di rappresentare numeri piccoli con intervalli minori fra loro rispetto ai numeri grandi
  - questo riduce gli errori nel calcolo a parità di bit utilizzati

# La rappresentazione dei numeri all'interno di un computer (11)

- **Rappresentazione in *virgola mobile* (cont.)**

- ogni numero  $N$  è rappresentato da una coppia  
(mantissa  $M$ , esponente  $E$ ) con il seguente significato

$$N = M * 2^E$$

- **esempi:**

1. in base 10, con 3 cifre per la mantissa e 2 cifre per l'esponente riesco a rappresentare

$$349\,000\,000\,000 = 3,49 * 10^{11}$$

con la coppia (3.49,11) perché  $M = 3.49$  ed  $E = 11$

# La rappresentazione dei numeri all'interno di un computer (12)

- Rappresentazione in **virgola mobile** (cont.)

- esempi:

- 2. in base 10, con 3 cifre per la mantissa e 2 per l'esponente riesco a rappresentare

$$0.000\ 000\ 002 = 2.0 * 10^{-9}$$

con la coppia (2.0,-9) perché  $M = 2.0$  ed  $E = -9$

- sia **0.000 000 002** che **349 000 000 000** non sono rappresentabili in virgola fissa usando solo 5 ( $3M+2E$ ) cifre decimali



# Lo standard IEEE 754

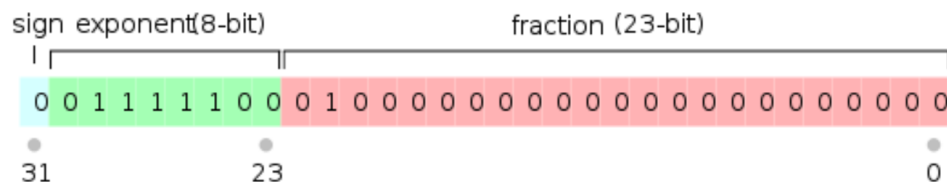
- Insieme di rappresentazioni di valori numerici e simbolici usato per ***floating point*** computation
  - Realizzabile attraverso software (librerie) o hardware
  - Vari tipi di precision
- **float** e **double** in C seguono questo standard
- Il numero viene formato con
  1. un bit di segno
  2. un esponente
  3. la mantissa

# Lo standard IEEE 754

- Si specificano 3 parametri:
  - **P**: precisione o numero di bit che compongono la mantissa
  - **E<sub>max</sub>**: esponente massimo
  - **E<sub>min</sub>**: esponente minimo

- Ad esempio per la **precisione singola (32 bit)**

- **P=23**, **E<sub>max</sub>=127** e **E<sub>min</sub>=-126**
- 1 bit *segno*; 8 bit *esponente*;



- La mantissa viene normalizzata scegliendo l'esponente in modo che sia sempre nella forma **1,xxxx...**
- L'esponente è polarizzato, ovvero ci si somma **E<sub>max</sub>**
  - costante di polarizzazione o **bias**

# Esempio

- $0.15625_{(10)} = \frac{1}{8} + \frac{1}{32} = 2^{-3} + 2^{-5} = 0.00101_{(2)}$
- $0.00101_{(2)} = 1.01_{(2)} \times 2^{-3}$  **Normalizzazione** della mantissa
  - Parte intera della mantissa (prima cifra) sempre diversa da zero
- In base 2 l'unica cifra diversa da 0 è 1 → posso non memorizzarla
  - **Parte frazionaria della mantissa:**  $.01_{(2)}$
  - **Esponente:** -3
  - **Esponente polarizzato** (precisione singola):  $-3 + 127 = 124$

# Esempio

- $0.15625_{(10)} = \frac{1}{8} + \frac{1}{32} = 2^{-3} + 2^{-5} = 0.00101_{(2)}$
- $0.00101_{(2)} = 1.01_{(2)} \times 2^{-3}$
- Per la **precisione doppia (64 bit)**
  - **P=52,  $E_{\max}=1023$  e  $E_{\min}=-1022$**  (1 bit *segno*; 11 bit *esponente*)
  - **Parte frazionaria della mantissa:**  $.01_{(2)}$
  - **Esponente:** -3
  - **Esponente polarizzato** (precisione singola):  $-3 + 1023 = 1020$

# Lo standard IEEE 754

- Per la precisione singola usare le seguenti relazioni:

	esp	M	numero
Numero normalizzato	$0 < \text{esp} < 255$	qualsiasi	$(-1)^s (1, M) 2^{\text{esp}-127}$
Numero denormalizzato	$\text{esp}=0$	$M \neq 0$	$(-1)^s (\underline{0}, M) 2^{-126}$ Riduce la perdita di precisione se underflow
Zero	$\text{esp}=0$	$M = 0$	$(-1)^s 0$
Infinito	$\text{esp}=255$	$M = 0$	$(-1)^s \infty$
NaN (Not a Number)	$\text{esp}=255$	$M \neq 0$	<i>NaN</i>

# Range e precisione

- **I numeri piu' piccoli (vicini allo zero) rappresentabili**

- Esp=1, M=0  $\rightarrow \pm 2^{-126} \approx \pm 1.17549 \times 10^{-38}$  (norm. **singola**)  
 $\pm 2^{-1022} \approx \pm 2.22507 \times 10^{-308}$  (norm. **doppia**)
- Esp=0, M=00...1  $\rightarrow \pm 2^{-149} \approx \pm 1.40130 \times 10^{-45}$  (denorm. **singola**)  
 $\pm 2^{-1074} \approx \pm 4.94066 \times 10^{-324}$  (denorm. **doppia**)

- **I numeri finiti piu' grandi (lontani dallo zero) rappresentabili sono**

- Esp=254, M=11...1  $\rightarrow \pm (1 - 2^{-24}) \times 2^{128} \approx \pm 3.40282 \times 10^{38}$  (prec. **singola**)  
 $\pm 1.79769 \times 10^{308}$  (prec. **doppia**)

# Esempio

- Voglio rappresentare il numero -36,47 usando la convenzione IEEE754, ovvero vedere come viene realmente memorizzata la variabile

```
float f=-36.47
```

- Prima di tutto calcolo la rappresentazione binaria. A tal fine calcolo parte intera e frazionaria mediante iterazione di divisioni/moltiplicazioni per 2.

# Esempio (2)

*(parte intera)*

36 div 2	(resto)
18	0
9	0
4	1
2	0
1	0
0	1

*(parte frazionaria)*

0,47 x 2	0	,94
0,94 x 2	1	,88
0,88 x 2	1	,76
0,76 x 2	1	,52
0,52 x 2	1	,04
0,04 x 2	0	,08
...	...	...

$$-36.47_{10} = -100100,011110..._2 = -1,0010001111 \times 2^5$$

Ora ho tutti gli elementi da collocare nella rappresentazione.



## Esempio (2)

Si continua finché la parte frazionaria è diversa da zero (o finché ci sono cifre nella rappresentazione...)

(parte intera)

36 div 2	(resto)
18	0
9	0
4	1
2	0
1	0
0	1

(parte frazionaria)

0,47 x 2	0	,94
0,94 x 2	1	,88
0,88 x 2	1	,76
0,76 x 2	1	,52
0,52 x 2	1	,04
0,04 x 2	0	,08
...	...	...

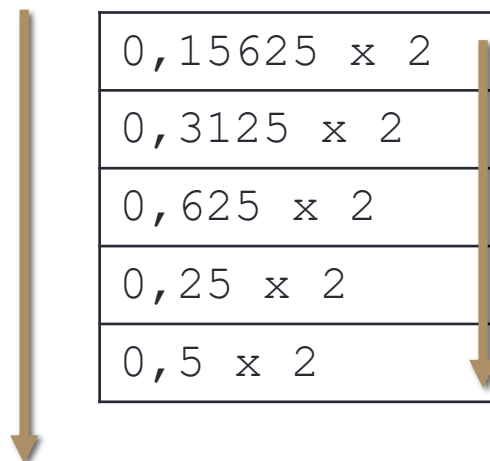
$$-36.47_{10} = -100100,011110..._2 = -1,0010001111 \times 2^5$$

Ora ho tutti gli elementi da collocare nella rappresentazione.

# Esempio (3)

- Rivediamo l'esempio di prima
- $0.15625_{(10)} = \frac{1}{8} + \frac{1}{32} = 2^{-3} + 2^{-5} = 0.00101_{(2)}$

*(parte frazionaria)*



0,15625 x 2	<b>0</b>	,3125
0,3125 x 2	<b>0</b>	,625
0,625 x 2	<b>1</b>	,25
0,25 x 2	<b>0</b>	,5
0,5 x 2	<b>1</b>	,00

# Un tool online per giocare con IEEE-754

- <https://www.h-schmidt.net/FloatConverter/IEEE754.html>

# Rappresentazione di un insieme finito di oggetti

- Vogliamo rappresentare i giorni della settimana :
  - $\{Lu, Ma, Me, Gio, Ve, Sa, Do\}$
  - usando sequenze 0 e 1
- Questo significa costruire un *codice*, cioè una tabella di corrispondenza che ad ogni giorno associa una opportuna sequenza
- In principio possiamo scegliere in modo del tutto arbitrario....

# Rappresentazione di un insieme finito di oggetti

- Una possibile codifica binaria per i giorni della settimana

Lunedì	0100010001
Martedì	001
Mercoledì	1100000
Giovedì	1
Venerdì	101010
Sabato	111111
Domenica	000001

# Rappresentazione di un insieme finito di oggetti

- **Problema:** la tabella di corrispondenza fra codifiche tutte di lunghezza diversa
  - spreco di memoria
  - devo capire come interpretare una sequenza di codifiche
  - 110000011 = Me Gio Gio
  - 110000011 = Gio Gio Do Gio
- **Soluzione:** si usa **un numero di bit uguale per tutti**, il minimo indispensabile

# Rappresentazione di un insieme finito di oggetti

- Per rappresentare 7 oggetti diversi servono almeno 3 bit (minima potenza di due che supera 7 è  $8 = 2^3$ ) quindi :
  - *000 Lunedì*
  - *001 Martedì*
  - *010 Mercoledì*
  - *011 Giovedì*
  - *100 Venerdì*
  - *101 Sabato*
  - *110 Domenica*
  - *111 non ammesso*

# Rappresentazione di caratteri e stringhe

- Tipologia di caratteri:
  - alfabeto e interpunzioni: A, B, ..., Z, a, b, ..., z, ,, :, “, ..
  - cifre e simboli matematici: 0, 1, ..., 9, +, -, >, ..
  - caratteri speciali: £, \$, %, ...
  - caratteri di controllo: CR, DEL, ....
- Le *stringhe* sono sequenze di caratteri terminate in modo particolare.
- I *caratteri* sono un insieme finito di oggetti e seguono la strategia vista per i giorni della settimana



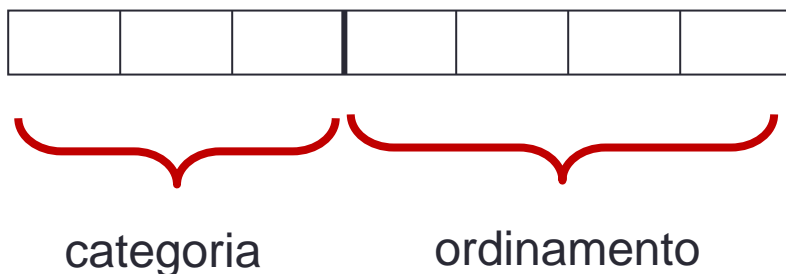
# Rappresentazione di caratteri e stringhe (2)

- **ASCII** (*American Standard Code for Information Interchange*):
  - Codice a 7 bit (standard)
- **ASCII** esteso a 8 bit (non standard)
  - es.: A           01000001  
          (           00101000
- **UNICODE**: su 16 bit (65536 diverse configurazioni): più recente, permette di rappresentare anche alfabeti diversi e simboli per la scrittura di lingua orientali.

# Rappresentazione di caratteri e stringhe (3)

- **ASCII a 7 bit**

I 7 bit sono suddivisi logicamente in due campi di 3 e 4 bit.



I primi tre bit rappresentano categorie di caratteri, mentre gli ultimi quattro servono a rispettare l'ordinamento dei caratteri all'interno di ogni categoria.

# Rappresentazione di caratteri e stringhe (4)


- **Categorie**

1°bit	2°bit	3°bit	Caratteri rappresentati
0	1	0	simboli di punteggiatura, simboli speciali e di operazione
0	1	1	numerali
1	0	0	maiuscole (A - O)
1	0	1	maiuscole (P - Z)
1	1	0	minuscole (a - o)
1	1	1	minuscole (p - z)

# Rappresentazione di caratteri e stringhe (5)

- 4°, 5°, 6°, 7° bit
- Nei numerali sono costituiti dalla codifica in binario su 4 bit delle cifre decimali (codice BCD).
- Per i caratteri dell'alfabeto la codifica è tale da rispettare l'ordinamento alfabetico

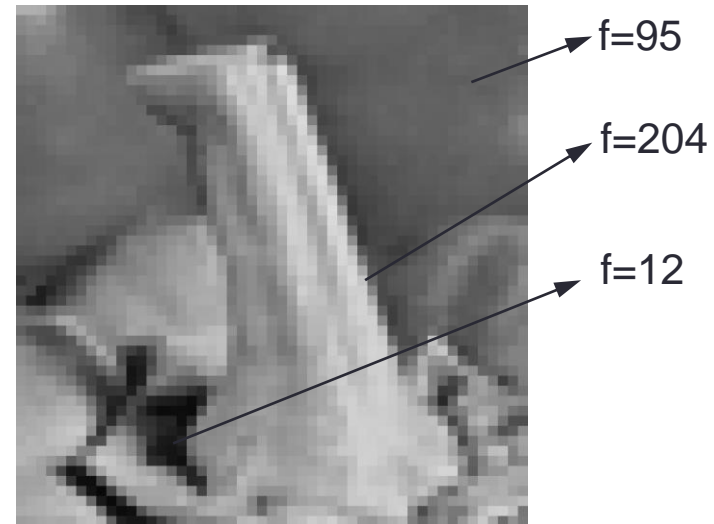
- **Esempi**

b (2<sup>a</sup> lettera) : 110 0010  


B: 100 0010

# Cos'è un'immagine digitale

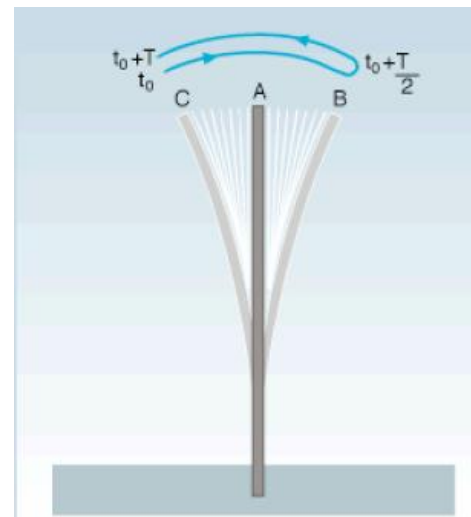
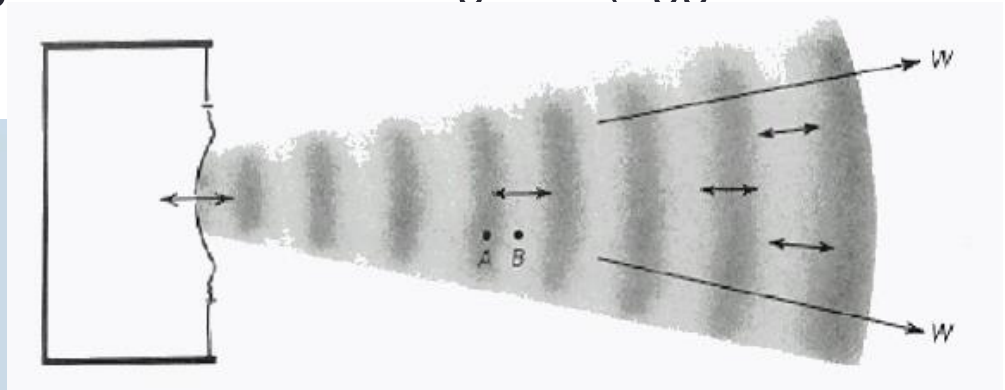
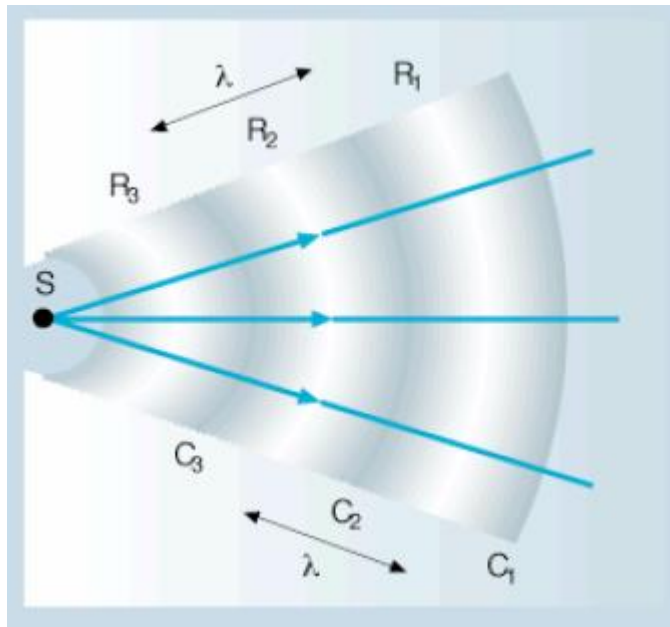
Un'**immagine digitale** può essere vista come una funzione bi-dimensionale  $f(x,y)$ , dove  $f$  rappresenta l'**intensità** o **livello di grigio** dell'immagine in quel punto: 0 rappresenta il **nero**, 255 il **bianco**



Un'*immagine* è quindi una **matrice** di elementi chiamati **pixels** (picture elements).

# La natura fisica del suono

- onde che trasportano energia lontano dalla sorgente (oggetto che vibra)



# Rappresentazione di audio

- E' possibile rappresentare il suono con la sua forma d'onda
- Al calcolatore, basta rappresentare una opportuna sequenza di campioni della forma d'onda

