Lezione 2

Rappresentazione dell'informazione

A. Borghese, F. Pedersini

Dip. Scienze dell'Informazione (DSI)
Università degli Studi di Milano

Terminologia

- **bit** = **b**inary dig**it**
- MULTIPLI binari:
  1 Byte (B) = 8 bit

1 kilobyte (kB)	= 2^{10} B	= 1024 B	
1 Megabyte (MB)	= 2^{20} B	= 1024 kB	= 1,048,576 B
1 Gigabyte (GB)	= 2^{30} B	= 1024 MB	= 1,073,741,824 B
1 Terabyte (TB)	= 2^{40} B	= 1024 GB	= 1,099,511,627,776 B
1 Exabyte (EB)	= 2^{50} B	= 1024 TB	= 1,125,899,906,842,624 B

- MULTIPLI decimali:
  - **1 kiB** = **10^3 byte** = 1000 **B**
  - **1 miB** = **10^6 byte** = 1000 **kiB** = 1.000.000 B
  - **1 giB** = **10^9 byte** = 1000 **miB** = 1.000.000.000 B
  - **1 tiB** = **10^{12} byte** = 1000 **giB** = 1.000.000.000.000 B
- Parola (word): numero di bit trattati come unicum dall'elaboratore
  - 8 bit Intel 8080, Z80
  - 16 bit Intel 8086
  - 32 bit MIPS, Intel Pentium
  - 64 bit Intel CoreDuo, IBM PowerPC G5, AMD Athlon64
  - 128 bit CELL processor (Sony Playstation III)

Definizione di **rappresentazione** di informazione:

- Corrispondenza tra informazione I e parola P(I) composta da un alfabeto A di simboli

$$\mathbf{I} \longrightarrow P(\mathbf{I}) = \{a_i\}, \quad a_i \in \mathbf{A}$$

- ALFABETO: Insieme dei simboli della rappresentazione
  - **A … Z**
  - **0 … 9**
  - **0 , 1**
  - I simboli dell'alfabeto possono assumere diverse forme
    - segni su carta, livelli di tensione, fori su carta, segnali di fumo…
  - Diversi alfabeti possono essere usati per rappresentare la stessa informazione

Capacità di codifica

- Quanti oggetti diversi (quanta informazione) riesco a rappresentare con 1 bit?
  - Con una parola di 1 bit rappresentiamo 2 oggetti
- Quanti oggetti posso rappresentare con k bit?
  - (2 x 2 x 2… x 2) = $\mathbf{2^k}$ oggetti
- *Quanti bit devono avere le parole binarie usate per identificare N oggetti diversi?*
  - *Es: N = 21: (A,B,…,Z)* $\quad 2^4 = 16 < 21 < 32 = 2^5 \rightarrow$ **5 bit**

  → Per N oggetti: *ceil($\log_2 N$)*
- *Quanti oggetti posso rappresentare con k cifre decimali?*
  - (10 x 10 x 10… x 10) = $\mathbf{10^k}$ oggetti
- In generale, con k cifre in base B rappresento $\mathbf{B^K}$ oggetti

Numerazione

- Se la rappresentazione è fatta con un alfabeto di numeri, si parla di numerazione
  - Numero elementi dell'alfabeto = **BASE**
- Numerazione araba DECIMALE
  - **S = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} – Base=10**
- Numerazione ESADECIMALE
  - **S = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F} – Base=16**
- Numerazione BINARIA:
  - **S = {0, 1} – Base=2**
- Numerazione OTTALE:
  - **S = {0, 7} – Base=8**

Numerazione posizionale

- Sistema di **conteggio**
  - Sistema di numerazione mediante simboli
  - Numerazione romana: **I, V, X, L, C, M**
  - La cifra "**X**" ha sempre lo stesso valore
- Sistema di **numerazione posizionale**
  - **cifra + peso**
  - Il peso è la base elevata alla posizione della cifra.
  - La cifra "**1**" ha un valore diverso nelle due scritture:
    - **100**
    - **1000**

Codifica posizionale di un numero

- Fondata sul concetto di base: $B_N = [b_0, b_1, b_2, b_3, \ldots]$
- Ciasun numero **E**, può essere rappresentato come combinazione lineare degli elementi della base:

$$E : \langle c_k \; c_{k-1} \cdots \; c_0 \rangle \qquad E = \sum_k c_k \cdot b_k$$

- Gli elementi della base sono i **PESI** delle cifre, in funzione:
  - della loro posizione i,
  - dell'ordine della base, n

$$\Rightarrow \qquad b_k = n^k$$

- Basi di numerazione:
  - $B_2 = \{\ldots 16, 8, 4, 2, 1, ½, ¼, \ldots\}$ base 2 (binaria)
  - $B_{10} = \{\ldots 1000, 100, 10, 1, 0.1, 0.01, \ldots\}$ base 10 (decimale)
  - $B_{16} = \{\ldots 4096, 256, 16, 1, 1/16, 1/256, \ldots\}$ base 16 (esadecimale)
  - Esempi:

$$12_{10} = 1 \cdot 10^1 + 2 \cdot 10^0$$
$$100_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 4_{10}$$

Conversione: *base 10 → base n*

- Un numero ***x*** in base 10 si trasforma in base *n* usando il seguente procedimento:
  - Dividere il numero ***x*** per ***n***
  - Il resto della divisione è la cifra di posto 0 in base *n*
  - Il quoziente della divisione è a sua volta diviso per *n*
  - Il **resto** ottenuto **è la cifra** di posto 1 in base *n*
  - Si prosegue con le divisioni dei quozienti ottenuti al passo precedente fino a che l'ultimo quoziente è 0.
  - l'ultimo resto è la cifra più significativa in base *n*

Conversione base 10 → base 2

Vogliamo rappresentare 1492_{10} *in binario:* $101\ 1101\ 0100_2$

1492 = 2 x 746 + 0 ← Bit meno significativo
746 = 2 x 373 + 0
373 = 2 x 186 + 1
186 = 2 x 93 + 0
93 = 2 x 46 + 1
46 = 2 x 23 + 0
23 = 2 x 11 + 1
11 = 2 x 5 + 1
5 = 2 x 2 + 1
2 = 2 x 1 + 0
1 = 2 x 0 + 1 ← Bit più significativo

Conversione: base n → base 10

Un numero a k cifre, in base *n*: E =<c_1 c_2 c_3 ...>
$b_i = n^i$

si trasforma in base 10, facendo riferimento alla formula:

$$E = \sum_{i=0}^{k-1} c_i \cdot b_i = \sum_{i=0}^{k-1} c_i \cdot n^i, \quad n = 10$$

Esempio:

$$101\ 1101\ 0100_2 = 1x2^{10} + 0x2^9 + 1x2^8 + 1x2^7 + 1x2^6 + 0x2^5 + 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0 =$$

$$1024 + 256 + 128 + 64 + 16 + 4 = 1492_{10}$$

Esercizi

- Dati i numeri decimali: 121331, 2453, 11101
  - si trasformino in base 3
  - si trasformino in base 7
  - si trasformino in base 2

- Dati i numeri:
  23456_7 , 121331_5 , 2453_8 , 111010101_2
  - convertire ciascuno in decimale e in binario

Codifica esadecimale

- La codifica esadecimale viene utilizzata come forma compatta per rappresentare numeri binari
  - 16 simboli: 0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F
    - Valore decimale: 0…15
  - $16 = 2^4 \rightarrow$ 1 cifra esadecimale = 4 bit

- Diverse notazioni equivalenti:
  - $9F_{16}$, **0x**9F , 9F**hex**

$$0x9F = 9 \cdot 16^1 + 15 \cdot 16^0 = 144 + 15 = 159_{10}$$

Conversione: esadecimale → binario

- Vogliamo rappresentare 9Fhex in base 2.
  - E' più semplice della conv. decimale
- Ogni cifra esadecimale viene convertita in un numero binario di 4 cifre:
  - 9 hex → 1001_2
  - F hex → 1111_2
  - 9F hex → $1001\ 1111_2$
- È sufficiente ricordarsi come si rappresentano in binario le singole cifre esadecimali da 0 a F

Conversione binario → esadecimale

- Ogni gruppo di 4 cifre viene tradotto nel simbolo corrispondente
  - Esempio: convertire 1101011_{due} in esadecimale:
  - 1011_{due} → B_{hex}
  - 0110_{due} → 6_{hex} (viene aggiunto un "leading" 0)
  - 1101011_{due} → $6B_{hex}$

0	1	2	3	4	5	6	7
0000	0001	0010	0011	0100	0101	0110	0111

8	9	A/10	B/11	C/12	D/13	E/14	F/15
1000	1001	1010	1011	1100	1101	1110	1111

Rappresentazione binaria di numeri negativi

- Codifica a **modulo e segno**: il primo bit indica il segno, il resto il numero in modulo.
- Codifica in **complemento a 1**: il numero negativo si ottiene cambiando 0 con 1 e viceversa.
- Svantaggi:
  - Ridondanti: doppia codifica per lo 0
  - Scomode per calcolo automatico
- Codifica in **complemento a 2**: il numero negativo si ottiene cambiando 0 con 1 e sommando 1.
  - Comodo per le operazioni (+/–)
  - La prima cifra rappresenta comunque il bit di segno

Modulo e segno	
dec	bin
0	00
+1	01
0	10
–1	11

Compl. a 1	
dec	bin
0	00
+1	01
–1	10
0	11

Compl. a 2	
dec	bin
0	00
+1	01
–2	10
–1	11

Operazioni tra numeri binari interi: ***somma***

- SOMMA
  - *Come le somme decimali fatte a mano, a scuola (2^ elementare)*

111 ← Riporti
1011 + 11_{10}
110 = 6_{10}

10001 17_{10}

Sottrazione fra interi

- Sfruttando i numeri negativi, gestisco la sottrazione come una somma:

  $11 - 13 = 11 + (-13)$

- Utilizzo la rappresentazione: complemento a 2

  $+11_{10} \rightarrow 01011_2$

  $-13_{10} \rightarrow 10011_2$

- Vantaggio della rappr. a complemento a 2

```
    11 ← riporti
 01011 +
 10011 =
-----------
 11110  → -2(10)
```


Moltiplicazione di interi

Come le moltiplicazioni decimali fatte a mano…

(3^ elementare)

```
27(10)   1 1 0 1 1  x
 7(10)       1 1 1 =
----------------------
 1 1 1 1 1 1
       1 1 0 1 1 +
     1 1 0 1 1 -
   1 1 0 1 1 - -
----------------------
 1 0 1 1 1 1 0 1   189(10)
```

Rappresentazione binaria di **numeri interi**:

N bit → 2^N valori rappresentabili

- Interi "unsigned" (senza segno)
  Da 0 (00…0) a $2^N - 1$ (111…1)
- Interi "signed"
  Da -2^{N-1} (100…0) a $2^{N-1} - 1$ (011…1)
  - Rappr. Complemento a 2

Es: MS Visual C++: Intero è su 4 byte (word di 32 bit):

signed int: $-2.147.483.650 \le c \le 2.147.483.649$

Algoritmo per la parte frazionaria

- Un numero ***x,y*** in base 10 si trasforma in base *n* usando il seguente procedimento:

- Per la parte frazionaria ***y***:
  - Moltiplicare il numero *y* per *n*
  - La prima cifra del risultato coincide con la prima cifra dopo la virgola.
  - Si elimina la parte intera e si considera la nuova parte frazionaria.
  - La parte frazionaria ottenuta viene moltiplicata per la base *n*.
  - La prima cifra del risultato è la seconda cifra dopo la virgola.
  - Si prosegue con le moltiplicazioni della parte frazionaria fino a quando:
    - Il resto diventa 0
    - si esaurisce la capacità di rappresentazione.

Conversione dei numeri decimali

Esempio:
$10{,}75_{10} = 1010{,}11_2$

$10 : 2 = 5, \mathbf{0}$ $\quad 0{,}75 \times 2 = 1.5 \rightarrow \mathbf{1}$
$5 : 2 = 2, \mathbf{1}$ $\quad 0{,}5 \times 2 = 1.0 \rightarrow \mathbf{1}$
$2 : 2 = 1, \mathbf{0}$
$1 : 2 = 0, \mathbf{1}$ $\quad \rightarrow \ldots,11$
(parte frazionaria)

$\rightarrow$ 1010,...
(parte intera)

Esempio:
$10{,}76_{10} = 1010{,}1100001\ldots_2$

$0{,}76 \times 2 = 1.52 \rightarrow \mathbf{1}$
$0{,}52 \times 2 = 1.04 \rightarrow \mathbf{1}$
$0.04 \times 2 = 0.08 \rightarrow \mathbf{0}$
$0.08 \times 2 = 0.16 \rightarrow \mathbf{0}$
$0.16 \times 2 = 0.32 \rightarrow \mathbf{0}$
$0.32 \times 2 = 0.64 \rightarrow \mathbf{0}$
$0.64 \times 2 = 1.28 \rightarrow \mathbf{1}$
$0.28\ldots?$

$\rightarrow$,1100001
errore $= 0.28 \cdot 2^{-8}$

- Errori di approssimazione:
  - arrotondamento e troncamento.

Moltiplicazione/divisione mediante ***shift***

- Lo shift di un numero a **destra**, di k cifre, corrisponde ad una divisione per la base elevata alla k-esima potenza
- Lo shift di un numero a **sinistra**, di k cifre, corrisponde ad una moltiplicazione per la base elev. alla k-esima potenza

$213_{10} / 10 = 21.3_{10}$

$$213_{10} / 10 = (2\cdot10^2 + 1\cdot10^1 + 3\cdot10^0) / 10 =$$
$$(2\cdot10^2 + 1\cdot10^1 + 3\cdot10^0) \cdot 10^{-1} =$$
$$(2\cdot10^1 + 1\cdot10^0 + 3\cdot10^{-1}) = 21.3 \quad \text{c.v.d.}$$

In binario:
$23_{10} = 10111_2 \rightarrow 101{,}11_2 = 23_{10} / 2^2 = 5,\ 75$

verifica:
$101.11_2 = 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 + 1\cdot2^{-1} + 1\cdot2^{-2} = 5{,}75_{10}$ c.v.d.

Rappresentazione di numeri decimali

- Problema della rappresentazione di numeri reali
  - Sono in numero finito (n. finito di bit)
- Dato un certo numero di bit per codificare il numero decimale, esistono due tipi di codifiche possibili:
- Rappresentazione a virgola fissa (fixed point)
  - La virgola è in posizione fissa all'interno della stringa
- Supponiamo di avere a disposizione 8 cifre:

  + 127,35 = + | 1 | 2 | 7 | 3 | 5 | 0 | 0

  – 187 = – | 1 | 8 | 7 | 0 | 0 | 0 | 0

  + 0,14567 = + | 0 | 0 | 0 | 1 | 4 | 5 | 6 … ??

Rappresentazione "floating point"

- Rappresentazione in virgola mobile (**floating point**)

  mantissa + esponente
- Esempio:
  - $127,35 = 12,735 \times 10^1 = 1,2735 \times 10^2 = \mathbf{0,12735 \times 10^3}$
    - rappresentazione normalizzata ↗
  - E' possibile rappresentare numeri grandi che piccoli
- Supponiamo di avere a disposizione 8 cifre:

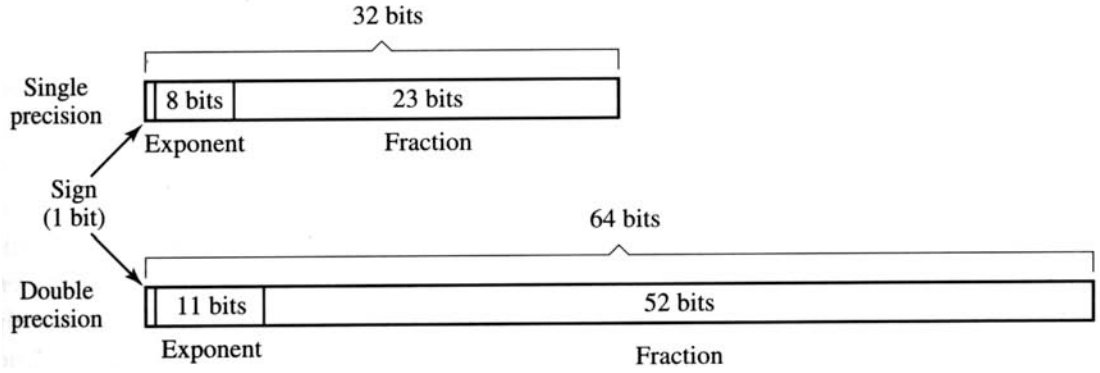
  6 cifre per la mantissa, 2 per l'esponente:

  + 127,35 = + | 1 | 2 | 7 | 3 | 5 | + | 3

  – 187 = – | 1 | 8 | 7 | 0 | 0 | + | 3

  + 0,14567 = + | 1 | 4 | 5 | 6 | 7 | + | 0

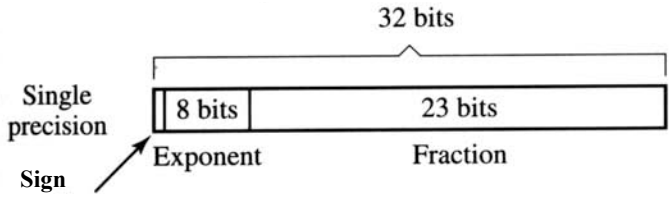
Lo standard: IEEE-754 (1980)



- Solo la parte frazionaria della mantissa
  - formato: 1,xxxxxxxx...
- Rappresentazione polarizzata dell'esponente:
  - **127** per singola precisione
  - 1 viene codificato come: 1000 0000.
  - **1023** in doppia precisione
  - 1 viene codificato come: 100 0000 0000.

Codifica standard IEEE-754

Single-precision:



Esempio: N = -10,75$_{10}$ = -1010,11$_2$

- Normalizzazione: ±1,xxxxxx $-1{,}01011 \times 2^3$
- Codifica del segno: 1 = "–" ; 0 = "+"
- Calcolo dell'esponente in rappresentazione polarizzata:

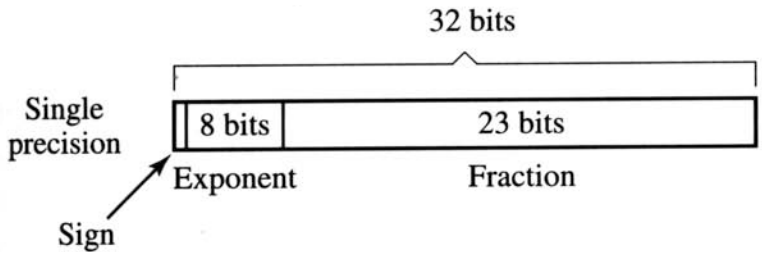
$$e = 3 + 127 = 130_{10} = 10000010_2$$

10.75_{10} =

1 ___8___ _____ 23 _____

= 1 | 1000 0010 | 01011000 00000000 0000000

IEEE 754: Configurazioni notevoli



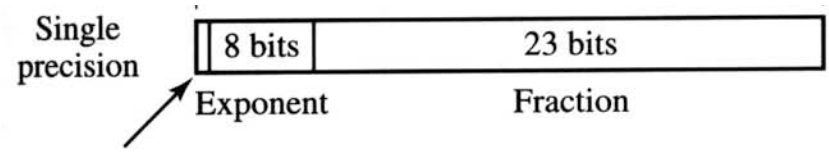
Numero	Mantissa	Esponente
0	= 0	0000 0000
∞	= 0	1111 1111
NaN	≠ 0	1111 1111
N. denormalizzato	≠ 0	0000 0000

Range esponenti: $1\ldots254 \rightarrow -126 \le e \le +127$

Range float (32 bit): $1{,}00\ldots0 \cdot 2^{-126} \le x \le 1{,}11\ldots1 \cdot 2^{+127}$

$1.175\ldots \cdot 10^{-38} \le x \le 3.4028\ldots \cdot 10^{+38}$

IEEE-754: numeri denormalizzati



MIN_float: $1{,}00\ldots00 \cdot 2^{-126} = 1.175\ldots \cdot 10^{-38}$

Float successivo: $1{,}00\ldots01 \cdot 2^{-126} = \text{MIN\_float} + 2^{-126-23}$

→ Risoluzione float: $2^{-126-23} = 2^{-149} = 1.41298\ldots \cdot 10^{-45}$

Discontinuità tra ZERO e MIN_float !!

- Soluzione: **numeri denormalizzati**

Non si procede alla normalizzazione:

Esponente: 00…0 (si assume che sia: −126)

Mantissa: $m_1 \ldots m_{23}$ → $\mathbf{0,}m_1\ldots m_{23} \cdot 2^{-126}$

→ MIN_denorm: $0{,}00\ldots01 \cdot 2^{-126} = 2^{-149} = 1.41298\ldots \cdot 10^{-45}$