MIDLAB Middleware Laboratory

Università di Roma "La Sapienza"
Dipartimento di Informatica e Sistemistica

Rappresentazione dell'informazione

Fondamenti di Informatica

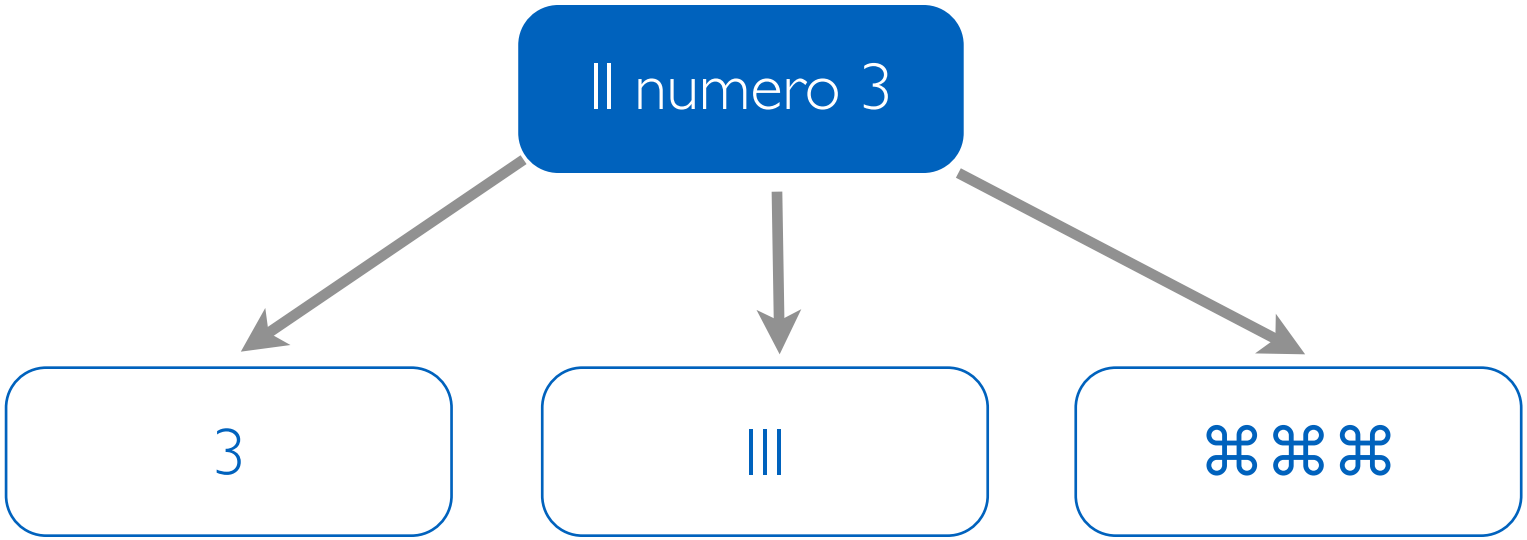
Ingegneria Gestionale

Leonardo Querzoni
querzoni@dis.uniroma1.it

A.A. 2010/2011

- *Informazione*: concetto astratto
- L'informazione deve essere rappresentata
- Utilizziamo un *supporto* per la rappresentazione
- Sul supporto vengono trascritti dei simboli
  - *codifica*: informazione ↦ simboli
  - *decodifica*: simboli ↦ informazione
- È necessario condividere un *codice*

- Come vengono rappresentati i numeri all'interno di un calcolatore?
- *Numero* = informazione astratta
- *Numerale* = stringa di caratteri che rappresenta un numero in un dato sistema di numerazione
- Uno stesso numero è rappresentabile con numerali differenti, a seconda del sistema di numerazione scelto
- Es:



- La quantità di numeri rappresentabili dipende dal numero di "cifre" (simboli) che compongono il numerale.
- Es: interi a 3 cifre con segno in notazione decimale [-999,+999]
- Se il numero di cifre è finito si perdono alcune proprietà:
  - Chiusura rispetto agli operatori
  - Proprietà associativa
- Es: interi a 2 cifre in notazione decimale [00,+99]
  - 78+36=?? (114) *overflow*
  - 20-30=?? (-10) *underflow*
  - 60+(50-40)≠(60+50)-40
- Se il numero di cifre è finito è impossibile rappresentare tutti i numeri reali.
- Si pone un problema di approssimazione

- Normalmente usiamo un sistema di numerazione *posizionale*
- In questi sistemi il numero rappresentato è dato dal valore di un polinomio i cui coefficienti (detti pesi) sono le potenze intere e positive di 10.
- Nei sistemi posizionali
  - ciascuna cifra rappresenta il coefficiente moltiplicativo di una potenza della base
  - l'esponente è dato dalla posizione del coefficiente nella stringa numerica
- Es: $133 = 1\times10^2 + 3\times10^1 + 3\times10^0 = 1\times100 + 3\times10 + 3\times1$
- Il sistema numerico Romano NON è posizionale

- A seconda della base utilizzata individuiamo diversi sistemi di numerazione:
  - B=10 decimale
  - B=2 binaria
  - B=8 ottale
  - B=16 esadecimale
- Se la base è B occorrono B simboli !
  - B=10 ↦ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  - B=2 ↦ {0, 1}
  - B=8 ↦ {0, 1, 2, 3, 4, 5, 6, 7}
  - B=16 ↦ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

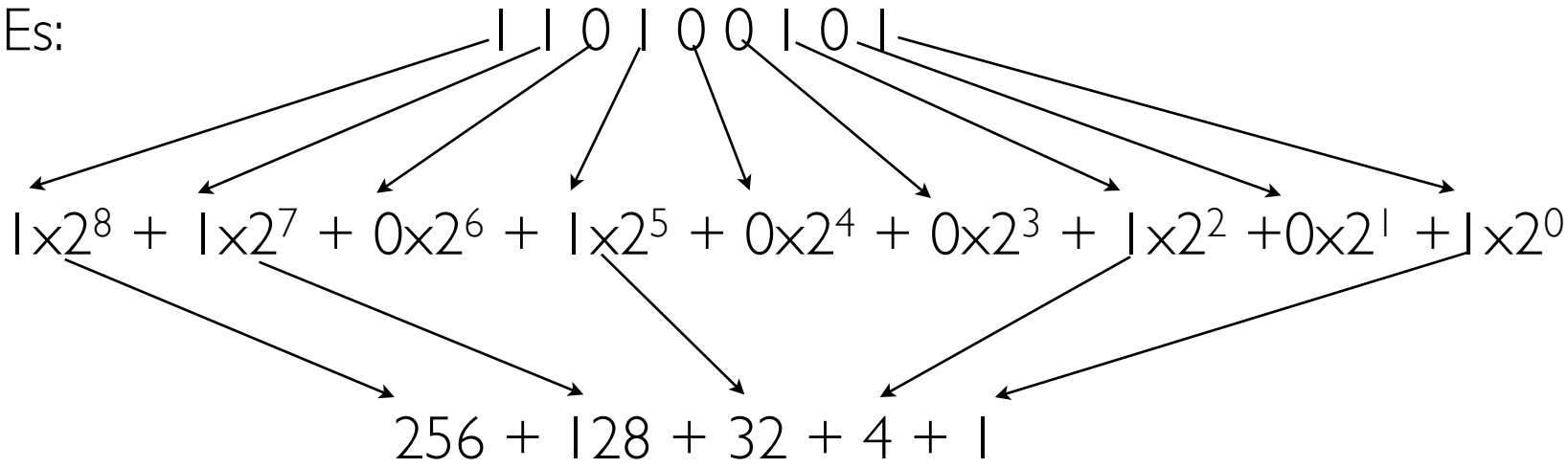
- I calcolatori elettronici moderni sono realizzati tramite elementi *bistabili* con cui è possibile rappresentare solo due simboli.
- Per questo motivo si utilizza il sistema di numerazione binario B=2
- ogni simbolo rappresenta un *bit* di informazione
- Si usano potenze di 2:
  - $2^0=1$
  - $2^1=2$
  - $2^2=4$
  - $2^3=8$
  - $2^4=16$
  - $2^5=32$
  - $2^6=64$
  - $2^7=128$
  - $2^8=256$
  - $2^9=512$
  - $2^{10}=1024$ Kilo
  - $2^{20}=1024$K Mega
  - $2^{30}=1024$M Giga
  - $2^{40}=1024$G Tera

- Conversione B=10 ↦ B=2
- Si divide per due il numero segnando quoziente e resto
- Ci si ferma quando il quoziente è 0

N	N/2	resto	cifra
25	12	1	1
12	6	0	0
6	3	0	0
3	1	1	1
1	0	1	1

- $(25)_{10} = (11001)_2$

- Conversione B=2 ↦ B=10
- Si moltiplica ogni coefficiente per la relativa potenza
- Si sommano i risultati
- Es:



421

- $(110100101)_2=(421)_{10}$

- Questi metodi di conversione sono generali e possono essere utilizzati per convertire un numero espresso con qualsiasi base da e verso la numerazione in base 10.
- Es: $(228)_{10} = (???)_{16}$

N	N/16	resto	cifra
228	14	4	4
14	0	14	E

$(228)_{10} = (E4)_{16}$

- Es: $(721)_8 \mapsto 7\times8^2 + 2\times8^1 + 1\times8^0 = 7\times64+16+1=(465)_{10}$

- Conversione diretta esadecimale ⇔ binario

F	5	7	A	3	1
1111	0101	0111	1010	0011	0001

- Come rappresentiamo i numeri negativi ?
- Esistono diversi metodi:
  - Rappresentazione in modulo e segno
  - Rappresentazione in complemento a 2
  - Rappresentazione con eccesso

Rappresentazione con modulo e segno

- Un bit dedicato al segno (1 = numero negativo)
- i restanti bit dedicati al modulo
- Es: 8 bit ↦ 1 bit + 7 bit ↦ [-127,+127]
- -9 ↦ |9| = $(0001001)_2$ ↦ -9 = $(1\text{-}0001001)_2$
- 9 = $(0\text{-}0001001)_2$

- Problemi:
  - Doppia rappresentazione dello 0 (+0 e -0)
  - Addizioni e sottrazioni non possono essere trattate dallo stesso circuito (il segno deve essere trattato a parte)

Rappresentazione in complemento a 2

- Immaginiamo di avere a disposizione n bit
- Il numero x è rappresentato come 2^n+x
- Permette di rappresentare i numeri $[-2^{n-1},+(2^{n-1}-1)]$

- Es: rappresentare x=-47 con n=8 bit e complemento a 2

  $2^n+x = 256-47 = 209 \mapsto (-47)_{10} = (11010001)_2$

- Un metodo semplice per la complementazione:
  - considero la rappresentazione binaria del modulo
  - partendo da destra lascio inalterati tutti i bit fino al primo "1" compreso
  - inverto tutti i bit seguenti

- Es: rappresentare x=-47 con n=8 bit e complemento a 2
- $(|x|)_{10} = (47)_{10} = (00101111)_2$

  00101111 rappresentazione binaria di partenza

  00101111 partendo da destra lascio inalterati tutti i bit fino al primo "1" compreso

  110100001 inverto tutti i bit che seguono

- $(-47)_{10} = (11010001)_2$

- Vantaggi
  - singola rappresentazione per lo 0
  - semplifica le operazione aritmetiche (ed i relativi circuiti !)

- La somma di due numeri è calcolata sempre nello stesso modo, indipendentemente dal loro segno:

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

- Es:

```
    1
  0 0 1 0 1 1 0 0 +        (44)10
  1 0 1 1 0 0 1 0 =       -(78)10
  ---------------------------------
  1 1 0 1 1 1 1 0         -(34)10

1 1   1 1 1
  1 1 0 1 0 1 0 0 +       -(44)10
  0 1 0 0 1 1 1 0 =        (78)10
  ---------------------------------
(1)0 0 1 0 0 0 1 0         (34)10
```

- Si può però verificare un fenomeno chiamato *overflow*: quando il risultato è troppo grande per essere rappresentato con i numeri a disposizione.
- Se:
  - i numeri sommati hanno segni diversi non si ha mai *overflow*
  - i numeri sommati hanno lo stesso segno si può verificare *overflow*
- In quest'ultimo caso il risultato avrà segno opposto a quello degli operandi.
- Es:

```
  1 1 1 1 1
  0 1 0 0 1 1 1 0 +        (78)10
  0 0 1 1 1 1 0 0 =        (60)10
  ---------------          -----
  1 0 0 0 1 0 1 0         -(118)10
```

Rappresentazione con eccesso

- Immaginiamo di avere a disposizione n bit
- Il numero x è rappresentato come $x+2^{n-1}$
- I numeri $[-2^{n-1},+(2^{n-1}-1)]$ sono rappresentati dall'intervallo $[0,2^n-1]$

- Es: rappresentare $x=-3$ con $n=8$ bit e con eccesso 2^{n-1}:

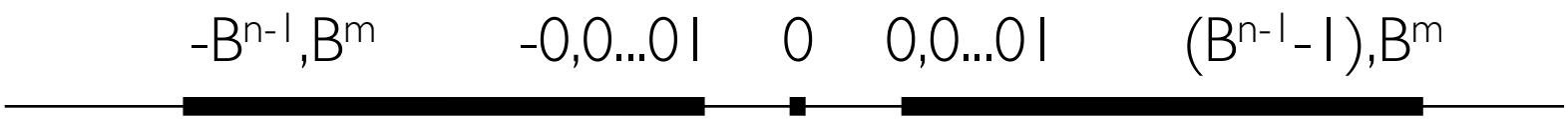
  $x+2^{n-1} = -3+128 = 125 \mapsto (-3)_{10} = (01111101)_2$

- Come possiamo rappresentare i numeri reali ?
- Qualunque sia la rappresentazione adottata, la quantità di simboli per numerale utilizzata è finita quindi:
  - L'intervallo dei numeri rappresentabili è limitato
  - All'interno di questo intervallo solo alcuni numeri sono rappresentabili
- Due modi per rappresentare i numeri reali:
  - Rappresentazione in virgola fissa
  - Rappresentazione in virgola mobile

- Con la rappresentazione in virgola fissa un certo numero di cifre è dedicato alla parte intera, ed altre sono dedicate alla parte decimale:

$$\underbrace{1285}_{n}\,,\,\underbrace{778}_{m}$$

- Permette di rappresentare l'intervallo dei reali compreso tra:



- La precisione massima raggiungibile è $1/B^m$

- Parte intera e parte decimale vengono convertite in due fasi distinte.

- Per convertire la parte decimale si procede in modo simile.
- Es: $(0.8125)_{10} \mapsto (???)_2$

N	2xN	trunc(2xN)	cifra
0.8125	1.625	1	1
0.625	1.25	1	1
0.25	0.5	0	0
0.5	1	1	1
0			

- $(0.8125)_{10} \mapsto (0.11101)_2$

- Es: $(0.11101)_2 \mapsto (???)_{10}$

$$1 \quad 1 \quad 0 \quad 1$$

$$1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$$

$$0.5 + 0.25 + 0.0625$$

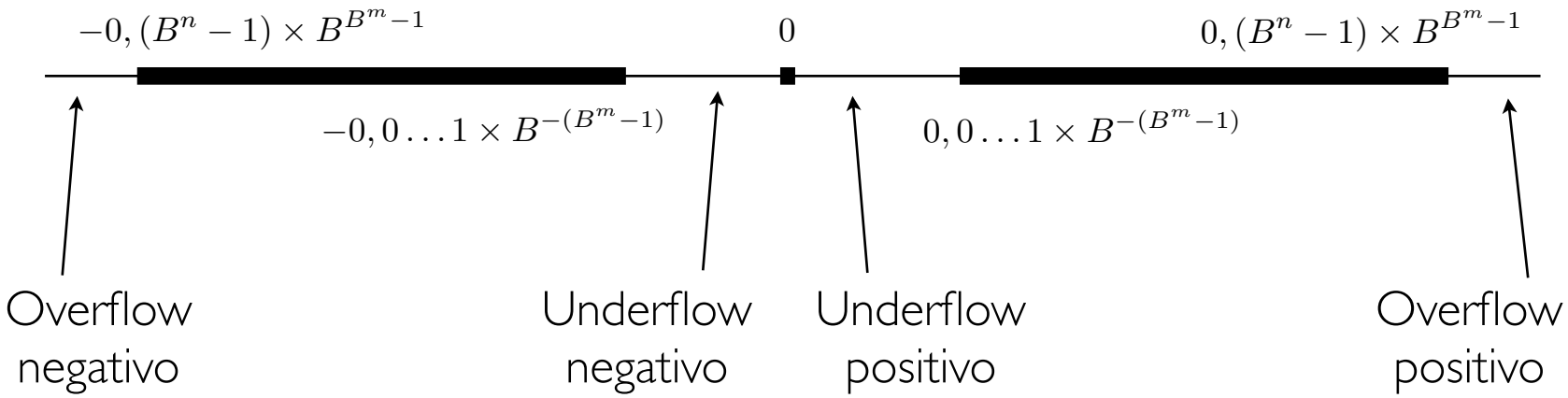
$$0.8125$$

$(0.11101)_2 \mapsto (0.8125)_{10}$

- La rappresentazione in virgola fissa obbliga a "sprecare" un certo numero di bit per la parte decimale (intera) anche se il numero da rappresentare non ne avrebbe bisogno.
- Per questo si adotta la rappresentazione in virgola mobile che si basa sulla *notazione scientifica*

$$1285{,}778 = 0{,}\underbrace{1285778}_{\text{mantissa (n)}} \times 10^{\overbrace{4}^{\text{esponente (m)}}}$$

- Permette di rappresentare l'intervallo dei reali compreso tra:



- Esempio in base 10:
  - n=5
  - m=3
  - più il segno sia per la mantissa che per l'esponente

- Posso rappresentare (oltre allo 0) i numeri compresi tra:
  - $-0{,}99999\times10^{999}$ e $-0{,}00001\times10^{-999}$
  - $0{,}00001\times10^{-999}$ e $0{,}99999\times10^{999}$

- Rispetto alla numerazione in virgola fissa posso:
  - rappresentare intervalli più grandi
  - aumentare la precisione dell'approssimazione

- Dato che la precisione della rappresentazione dipende dalla quantità di numerali (bit nel caso di nostro interesse) dedicati a mantissa ed esponente, sorge la necessità di uno standard:
- IEEE-754
  - numeri a *precisione singola*: 32bit
    - 1 bit per il segno
    - 8 bit per l'esponente (espresso con eccesso)
    - 23 bit per la mantissa
    - permette la rappresentazione di numeri compresi tra:
      - -10^{38} e -10^{-45}
      - 10^{-45} e 10^{38}
    - con una precisione di circa 7 cifre decimali
  - numeri in *precisione doppia*: 64bit
    - 1 bit per il segno
    - 11 bit per l'esponente
    - 52 bit per la mantissa

- Esempio: convertire il seguente numero espresso nel formato IEEE-754 a singola precisione in un numero decimale.

  0-01111110-10100000000000… … ……00000000000

  Il segno è positivo.

  Calcolo l'esponente:

  $(01111110) = 0+2^6+2^5+2^4+2^3+2^2+2^1 = 126$

  Devo rimuovere l'eccesso !

  esponente = 126-128 = -2

  La parte frazionaria della mantissa è $(.101)_2$

  Il numero è:

  $X = (-1)^0 \times (1.101)_2 \times 2^{-2} = (1+1/2+1/8)_{10} \times 2^{-2} =$

  $(1+0.5+0.125) \times 0.25 = (1.625) \times 0.25 = 0.40625$