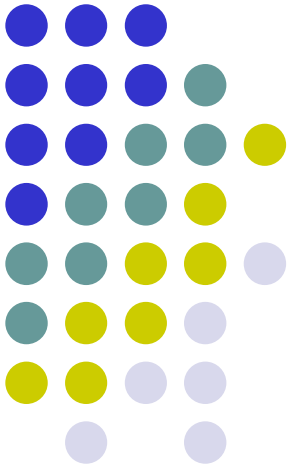
Laboratorio di Informatica

Corso di Laurea in Matematica

A.A. 2007/2008



Dott.Davide Di Ruscio

Dipartimento di Informatica

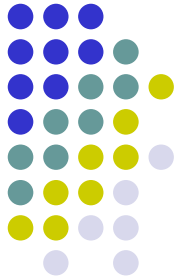
Università degli Studi di L'Aquila

Lezione del 07/12/07

Nota

Questi lucidi sono tratti dal materiale distribuito dalla McGraw-Hill e basati su del materiale fornito dal Prof. Flammini Michele

Rappresentazione dell'informazione (1/2)



Codifica binaria dell'informazione: modalità di rappresentazione dei dati (numeri, caratteri, immagini, …) in un elaboratore

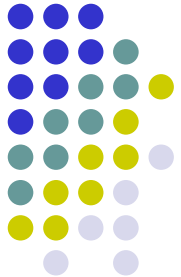
- Bit:
  - può avere soltanto valore 0 o 1
  - è l'unità elementare di informazione memorizzabile in un elaboratore
  - il suo valore corrisponde ad un possibile stato di un dispositivo bistabile (es., interruttore aperto o chiuso, cerchietto di semiconduttore magnetizzato o smagnetizzato, presenza o assenza di corrente o tensione, alternanza fra luce e buio nella trasmissione di dati su fibra ottica, ecc….)

- Byte = 8 bit.
- Kbyte = 2^{10} byte = 1024 byte
- Mbyte = $2^{10} \times 2^{10} = 2^{20}$ byte
- Gbyte = 2^{30} byte
- Tbyte = 2^{40} byte

- Es: una RAM di 512 Mbyte contiene 512×2^{20} byte o $512 \times 2^{20} \times 8$ bit

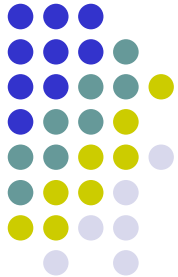
- Poiché $2^{10} \approx 1000$, a volte si approssimano K con mille, M con milione, G con miliardo, …

Rappresentazione dell'informazione (2/2)



- Internamente ad un elaboratore tutto viene codificato con sequenze di bit in modo trasparente all'utente, poiché il procedimento inverso di presentazione delle informazioni secondo i nostri formalismi e percezioni è effettuato in modo automatico dai dispositivi di Input/Output
- Un bit permette di distinguere tra loro solo due elementi o possibilità, rappresentate o codificate rispettivamente dai valori *0* ed *1*
- Per codificare un numero superiore di elementi, è necessario utilizzare sequenze di bit, il cui numero aumenta all'aumentare della loro lunghezza
- Il numero di sequenze di lunghezza *k* è dato dal numero di possibili configurazioni ottenibili assegnando valori nell'ordine ai *k* bit, ossia 2^k

Codifica binaria (1/2)

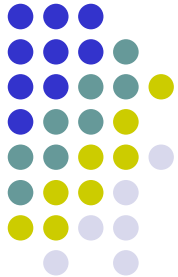


- Dato un insieme *U* di *n* elementi ed un numero intero *k* fissato, una codifica binaria di *U* a *k* bit consiste nell'assegnamento di sequenze di *k* bit agli elementi di *U* in modo univoco, ossia in modo tale che ad ogni elemento corrisponda un'unica sequenza e ad ogni sequenza un unico elemento
- Più formalmente una codifica è una funzione iniettiva $c:U \rightarrow \{0,1\}^k$

Esempio

- *U={Luca, Marco, Alessio}*
- Codifica di *U* a *2* bit: *c(Luca)=00, c(Marco)=01* e *c(Alessio)=10*

Codifica binaria (2/2)



- Domanda: quanto deve essere grande *k* per codificare un insieme *U* di *n* elementi?

- Con *k=1* si hanno soltanto *2* sequenze diverse, ossia *0* e *1*, per cui si possono codificare insiemi *U* con *n≤2*, ossia contenenti al più *2* elementi.
- Con *k=2* si hanno soltanto *4* sequenze diverse, ossia *00, 01, 10* e *11*, per cui si possono codificare insiemi *U* con *n≤4*
- Con *k=3* si hanno soltanto *8* sequenze diverse, per cui *n≤8*
- …
- Per un generico *k* deve essere $n \leq 2^k$, *ossia* $k \geq \log_2 n$

- In generale si cerca di scegliere *k* più piccolo possibile, in modo da risparmiare memoria
- Poiché *k* è un numero intero, il più piccolo valore possibile è $k = \lceil \log_2 n \rceil$

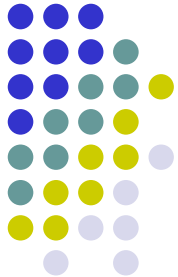
Riepilogando ...

n	*min k*
2	*1*
3	*2*
4	*2*
5	*3*
6	*3*
7	*3*
8	*3*
9	*4*
…	*…*

n	*min k*
16	*4*
17	*5*
…	*…*
32	*5*
33	*6*
…	*…*
64	*6*
65	*7*
…	*..*

n	*min k*
128	*7*
129	*8*
…	*…*
256	*8*
257	*9*
…	*…*
512	*9*
513	*10*
…	*…*

Codifica caratteri



Includono

- caratteri alfanumerici (lettere e cifre)
- simboli (punteggiatura, operatori aritmetici, …)
- caratteri di controllo (per formattazione, stampe, codici di trasmissione,…)

Alcuni codici noti: ASCII, EBCDIC, FIELDDATA, …

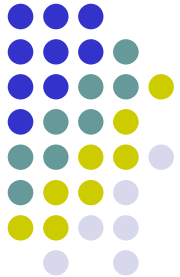
ASCII (American Standard Code for Information Interchange):

- 7 bit per carattere
- 1 bit per controllo errore (bit di parità, assegnato in modo da avere un numero pari di bit uguali ad 1 in tutti il byte)
- 128 caratteri rappresentati

Extended ASCII:

- 8 bit per carattere
- 256 caratteri rappresentati

Codifica numeri

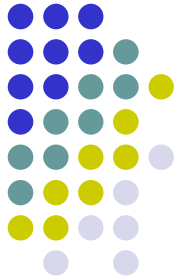


Codifiche per

- naturali (interi non negativi)
- interi (anche negativi)
- frazionari (reali compresi tra 0 e 1)
- reali

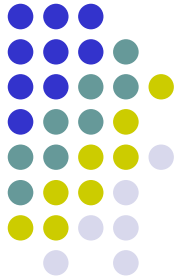
Le codifiche dei numeri non sono arbitrarie, ma progettate per facilitare lo svolgimento delle operazioni aritmetiche e garantire diversi livelli di precisione

Sistemi di numerazione (1/4)



- Il sistema di numerazione comunemente usato è *l'arabico* che rappresenta I numeri naturali tramite sequenze di cifre
- E' un sistema di numerazione in base dieci (le cifre usate sono dieci {0,...9})
- E' un sistema di numerazione posizionale, perché il significato attribuito a ciascuna cifra è funzione della posizione che tali cifra occupa nel numero

Sistemi di numerazione (2/4)



Sistema di numerazione posizionale in base *10*:

- utilizza *10* cifre (*0, …, 9*)
- il valore di ogni cifra dipende dalla posizione all'interno del numero
  Es. $357=7+5\cdot 10+3\cdot 100$
- in generale

$$a_{n-1}a_{n-2}\ldots a_1a_0=$$

$$=a_0\cdot 10^0+a_1\cdot 10^1+\ldots+a_{n-2}\cdot 10^{n-2}+a_{n-1}\cdot 10^{n-1}=$$

$$=\sum_{i=0}^{n-1} a_i\cdot 10^i \qquad a_i\in\{0,\ldots,9\}$$

È possibile estendere a qualsiasi base $b>0$

Sistemi di numerazione (3/4)

Sistema di numerazione posizionale in base *b*:

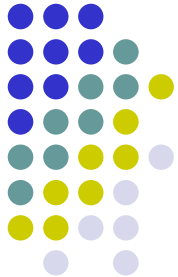
- utilizza *b* cifre
- il valore di ogni cifra dipende dalla posizione all'interno del numero:

$$(a_{n-1}a_{n-2}\ldots a_1 a_0)_b =$$

$$= a_0 \cdot b^0 + a_1 \cdot b^1 + \ldots + a_{n-2} \cdot b^{n-2} + a_{n-1} \cdot b^{n-1} =$$

$$= \sum_{i=0}^{n-1} a_i \cdot b^i \qquad a_i \in \{0,\ldots,b\text{-}1\}$$

Sistemi di numerazione (4/4)



Quando la base *b* è maggiore di *10*, per indicare le cifre superiori a *9* per convenzione si utilizzano le lettere maiuscole dell'alfabeto

Esempio: *b=16* (sistema esadecimale)

Cifra in base *16*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
Valore in base *10*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*

Esempi

$10011_2 =$

$= 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 = 1 + 2 + 16 = 19$

$237_8 =$

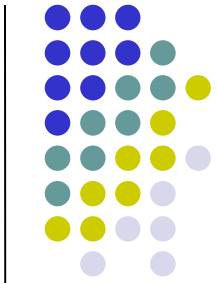
$= 7 \cdot 8^0 + 3 \cdot 8^1 + 2 \cdot 8^2 = 7 + 24 + 128 = 159$

$1AF_{16} =$

$= 15 \cdot 16^0 + 10 \cdot 16^1 + 1 \cdot 16^2 =$

$= 15 + 160 + 256 = 431$

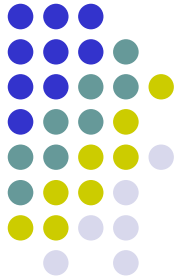
Conversioni di base



- Come convertire un numero in base *b* nel suo equivalente in una base *b'* ≠ *b*?

- Vedremo ora due regole generali, la cui distinzione fondamentale consiste nel fatto che effettuano le operazioni aritmetiche coinvolte rispettivamente nella base partenza *b* e nella base di arrivo *b'*

Regola 1

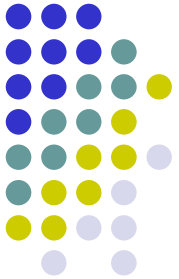


- Svolge le operazioni nella base di arrivo *b'*, per cui è molto adatta al caso in cui *b'=10*
- Consiste nell'applicare in modo diretto la sommatoria

$$(a_{n-1}a_{n-2}\ldots a_1 a_0)_b = \sum a_i \cdot b^i \qquad (1)$$

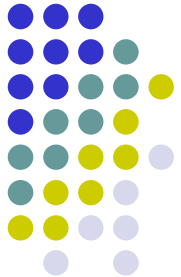
Regola 1

1. si esprimono le cifre a_i e la base *b* nella base *b'* (solitamente banale)
2. si calcola la sommatoria *(1)*



0000_2	=	0_{10}	=	0_8	=	0_{16}
0001_2	=	1_{10}	=	1_8	=	1_{16}
0010_2	=	2_{10}	=	2_8	=	2_{16}
0011_2	=	3_{10}	=	3_8	=	3_{16}
0100_2	=	4_{10}	=	4_8	=	4_{16}
0101_2	=	5_{10}	=	5_8	=	5_{16}
0110_2	=	6_{10}	=	6_8	=	6_{16}
0111_2	=	7_{10}	=	7_8	=	7_{16}
1000_2	=	8_{10}	=	10_8	=	8_{16}
1001_2	=	9_{10}	=	11_8	=	9_{16}
1010_2	=	10_{10}	=	12_8	=	A_{16}
1011_2	=	11_{10}	=	13_8	=	B_{16}
1100_2	=	12_{10}	=	14_8	=	C_{16}
1101_2	=	13_{10}	=	15_8	=	D_{16}
1110_2	=	14_{10}	=	16_8	=	E_{16}
1111_2	=	15_{10}	=	17_8	=	F_{16}

Regola 2
(delle divisioni successive)

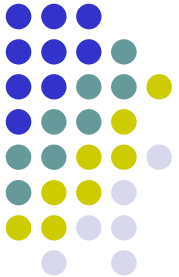


- Svolge le operazioni nella base di partenza *b,* per cui è molto adatta al caso in cui *b=10*

- Si basa sull'osservazione che, dividendo il numero per *b':*
  - il resto della divisione corrisponde alla cifra meno significativa del numero nella base *b'*
  - il quoziente della divisione corrisponde al numero ottenuto cancellando la cifra meno significativa dal numero di partenza espresso in base *b'*
  - le cifre più significative possono essere quindi determinate riapplicando ricorsivamente lo stesso metodo al quoziente

Esempio

357:10 è pari a *35* con resto *7*
35:10 è pari a *3* con resto *5*
3:10 è pari a *0* con resto *3*



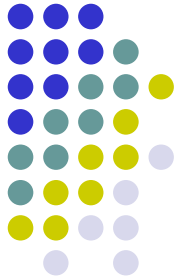
Regola 2:

- si determinano il quoziente ed il resto della divisione del numero per *b'*
- si prosegue come al passo *1*. considerando di volta in volta come numero di partenza il quoziente della divisione effettuata nel passo precedente, finché non si determina un quoziente nullo
- si scrivono tutti i resti ottenuti in ordine inverso, esprimendoli nella base *b'* (solitamente banale)

NB:

- il primo resto ottenuto (al passo 1.) corrisponde alla cifra meno significativa, mentre l'ultimo a quella più significativa
- se $b' < b$ tutti i resti ottenuti sono già espressi nella base *b'*

Esempi (1/3)



1) *37 (b=10 e b'=2)*

37:2 = 18 con resto 1
18:2 = 9 con resto 0
9:2 = 4 con resto 1
4:2 = 2 con resto 0
2:2 = 1 con resto 0
*1:2 = **0** con resto 1*

Quindi *37*=100101_2

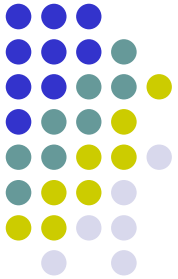
2) *3226 (b=10 e b'=16)*

3226:16 = 201 con resto 10 (A)
201:16 = 12 con resto 9 (9)
*12:16 = **0** con resto 12 (C)*

Quindi *3226*=$C9A_{16}$

Controprova:
$1x2^5 + 0x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 37_{10}$

Esempi (2/3)



3) *331 (b=10 e b'=2)*

331:2 = 165 con resto 1
165:2 = 82 con resto 1
82:2 = 41 con resto 0
41:2 = 20 con resto 1
20:2 = 10 con resto 0
10:2 = 5 con resto 0
5:2 = 2 con resto 1
2:2 = 1 con resto 0
*1:2 = **0** con resto 1*

Quindi $331_{10} = 101001011_2$

4) *331 (b=10 e b'=8)*

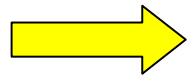
331:8 = 41 con resto 3
41:8 = 5 con resto 1
*5:8 = **0** con resto 5*

Quindi $331_{10} = 513_8$

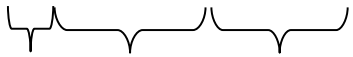
- Le BASI, 8 e 16 sono potenze di 2 ($8=2^3$ e $16=2^4$)
- Data la configurazione binaria *c* raggruppare le cifre in pacchetti di 3 (base 8) o in pacchetti di 4 (base 16) partendo da destra
- Esempio:

1010010011_2

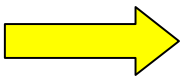
5 1 3



$1010010011_2 = 513_8$

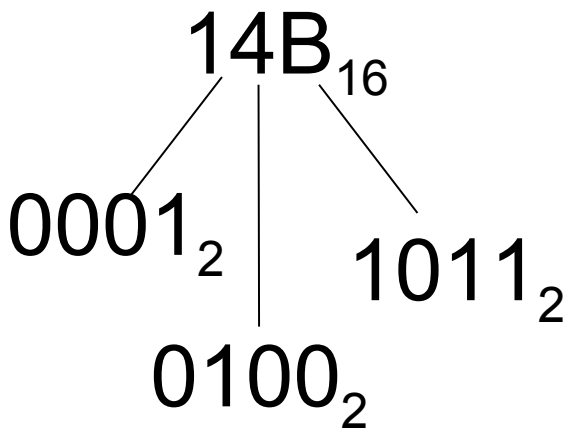


$1_{10}=1_{16}$ $4_{10}=4_{16}$ $11_{10}=B_{16}$



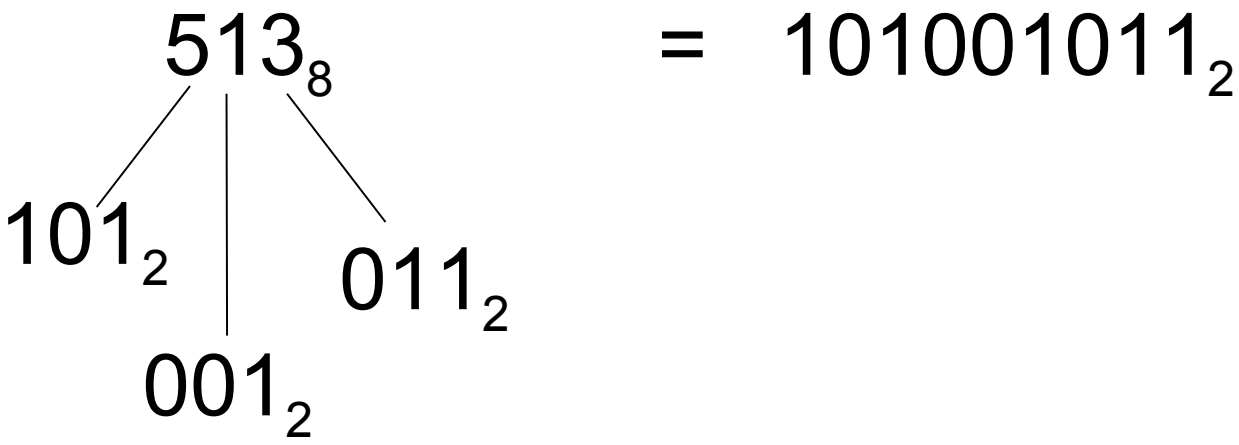
$1010010011_2 = 14B_{16}$

- Se abbiamo la configurazione $14B_{16}$ e vogliamo sapere quella binaria rappresento ogni cifra esadecimale con 4 bit

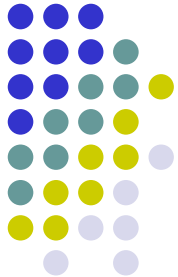


$= \quad 1010010112$

- Se abbiamo la configurazione 513_8 e vogliamo sapere quella binaria rappresento ogni cifra esadecimale con 3 bit



Codifica numeri naturali



- Poiché l'unità elementare di informazione in un elaboratore è il bit, che corrisponde alle due cifre *0* e *1,* in modo naturale viene utilizzato il sistema di numerazione posizionale in base *2*

- Fissato il numero di bit *k* da utilizzare nella rappresentazione
  1. si converte il numero di partenza nella base *2*
  2. si antepongono bit uguali a *0* al numero determinato fino ad ottenere complessivamente esattamente *k* bit

  NB: l'aggiunta dei bit pari a *0* nel passo 2. è necessaria perché nella memorizzazione del numero bisogna specificare per ogni bit (anche per i più significativi che convenzionalmente non indichiamo quando pari a *0*) lo stato del relativo dispositivo bistabile; in caso contrario si potrebbero avere errori di rappresentazione.

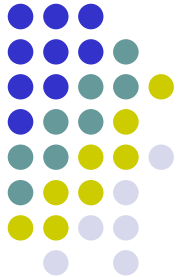
Esempio

- *43 (con k=8 e b'=2)*

43:2 = 21 con resto 1
21:2 = 10 con resto 1
10:2 = 5 con resto 0
5:2 = 2 con resto 1
2:2 = 1 con resto 0
*1:2 = **0** con resto 1*

Quindi $43=101011_2$, la cui rappresentazione a *8* bit è

00101011

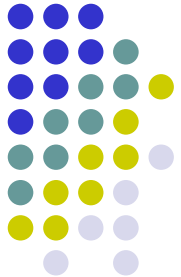


- Chiaramente, fissato il numero di bit *k*, è possibile rapprentare al più 2^k numeri interi, che vanno da *0 (0…0)* a 2^k*-1 (1…1)*

- Quindi, l'intervallo dei numeri rappresentabile con *k* bit è $[0,2^k-1]$

- Overflow: errore che si verifica quando si tenta di rappresentare un numero al di fuori dell'intervallo, ad esempio quando il risultato di un'operazione aritmetica è troppo grande

Codifica numeri frazionari (reali compresi tra *0* ed *1*)



- I numeri frazionari sono reali N t.c. $0 < N < 1$
- Non sono rappresentabili in maniera precisa in quanto per farlo servirebbero un numero infinito di cifre (e quindi di bit)

- Estendendo quanto detto per i sistemi di numerazione alla parte del numero dopo la virgola si ha che

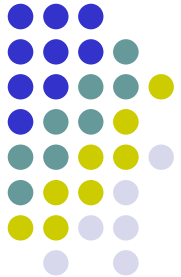
Es. $0{,}587_{10}$ = ($5 \times 10^{-1} + 8 \times 10^{-2} + 7 \times 10^{-3}$)

$$N_{10} = 0, a_{-1} a_{-2} \ldots a_{-n} = a_{-1} \cdot 10^{-1} + a_{-2} \cdot 10^{-2} + \ldots + a_{-n} \cdot 10^{-n} = \sum_{i=-1}^{-n} a_i \cdot 10^i$$

- Per una generica base *b*

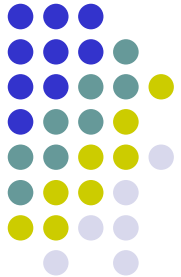
$$N_b = (0, a_{-1} \ldots a_{-n})_b = a_{-1} \cdot b^{-1} + \ldots + a_{-n} \cdot b^{-n} = \sum_{i=-1}^{-n} a_i \cdot b^i \qquad (2)$$

Il peso delle cifre a_i dipende dalla base prescelta nel sistema di numerazione



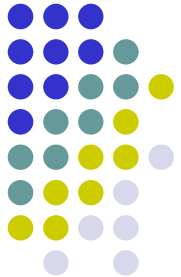
- Come convertire un numero frazionario in base *b* nel suo equivalente in una base *b'≠ b*?

- Di nuovo abbiamo due regole generali che effettuano le operazioni aritmetiche coinvolte rispettivamente nella base partenza *b* e nella base di arrivo *b'*



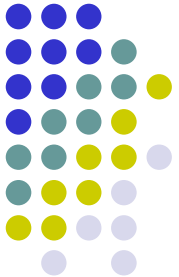
Regola 1:

- svolge le operazioni nella base di arrivo *b'*, per cui è molto adatta al caso in cui *b'=10*
- consiste nell'applicare in modo diretto la sommatoria *(2)* nel seguente modo:
  1. si esprimono le cifre a_i e la base *b* nella base *b'* (solitamente banale)
  2. si calcola la sommatoria *(2)*



Esempi

- $0,1101_2 =$
  $=1\cdot 2^{-1}+1\cdot 2^{-2}+0\cdot 2^{-3}+1\cdot 2^{-4}=$
  $=1/2 + 1/4 + 1/16=0.8125_{10}$
- $0,1011_2 =$
  $= 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4}=$
  $= 1/2 + 1/8 + 1/16 = 0,6875_{10}$
- $0,452_8 =$
  $= 4\cdot 8^{-1}+5\cdot 8^{-2}+2\cdot 8^{-3}=$
  $= 4/8+5/64+2/512=0,582031252_{10}$
- $0,1A8_{16} =$
  $= 1\cdot 16^{-1}+10\cdot 16^{-2}+8\cdot 16^{-3}=$
  $=1/16+10/256+8/4096=0,103515625_{10}$



Regola 2 (delle moltiplicazioni successive):

- svolge le operazione nella base di partenza *b,* per cui è molto adatta al caso in cui *b=10*

- si basa sull'osservazione che, moltiplicando il numero per *b':*
  - la parte intera corrisponde alla prima cifra dopo la virgola del numero nella base *b'*
  - la parte frazionaria al numero ottenuto cancellando la prima cifra dopo la virgola dal numero di partenza espresso in base *b'*
  - le cifre successive alla prima dopo la virgola possono essere quindi determinate riapplicando ricorsivamente lo stesso metodo alla parte frazionaria

Esempio

0.623·10 = 6.23, ossia parte intera 6 e frazionaria 0.23
0.23·10 = 2.3, ossia parte intera 2 e frazionaria 0. 3
0.3·10 = 3, ossia parte intera 3 e frazionaria 0

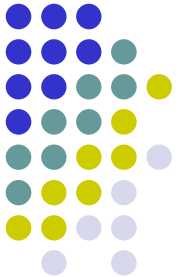
Regola 2:

1. si determinano la parte intera a frazionaria della moltiplicazione del numero per *b'*
2. si prosegue come al passo *1*. considerando di volta in volta come numero di partenza la parte frazionaria della moltiplicazione effettuata nel passo precedente, finché non si determina una parte frazionaria nulla
3. si scrivono tutte le parti intere nell'ordine in cui sono state ottenute, esprimendole nella base *b'* (solitamente banale)

NB:

- la prima parte intera ottenuta (al passo 1.) corrisponde alla prima cifra dopo la virgola, mentre l'ultima a quella più lontana
- se b'<b tutte le parti intere ottenute sono già espresse nella base *b'*

Esempi



1) 0.375 (b=10 e b'=2)

$0.375 \cdot 2 = 0 + 0.75$
$0.75 \cdot 2 = 1 + 0.5$
$0.5 \cdot 2 = 1 + \mathbf{0}$

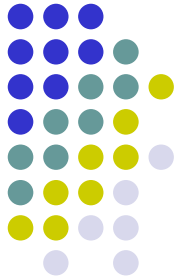
Quindi $0.375 = 0{,}011_2$

2) 0.84375 (b=10 e b'=16)

$0.84375 \cdot 16 = 13\ (D) + 0.5$
$0.5 \cdot 16 = 8\ (8) + \mathbf{0}$

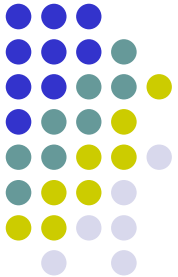
Quindi $0.84375 = 0{,}D8_{16}$

Nota



- I numeri frazionari possono introdurre approssimazioni dovute alla presenza di un numero limitato di cifre dopo la virgola
- L'approssimazione è comunque inferiore a b^{-n} dove n è il numero di cifre utilizzate

Esempio 1



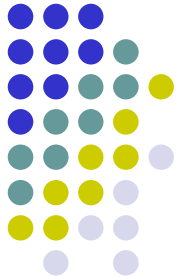
1) 0.587 (b=10 e b'=2)

$0.587 \cdot 2 = 1 + 0.174$
$0.174 \cdot 2 = 0 + 0.348$
$0.348 \cdot 2 = 0 + 0.696$
$0.696 \cdot 2 = 1 + 0.392$
$0.392 \cdot 2 = 0 + 0.784$
$0.784 \cdot 2 = 1 + 0.568$
$0.568 \cdot 2 = 1 + 0.136$
…

$0.587_{10} = 0.1001_2$ (approssimazione $< 2^{-4}$)

$0.587_{10} = 0.1001011_2$ (approssimazione $< 2^{-7}$)

Esempio 2



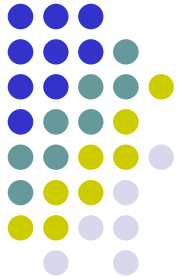
2) 0.35 (b=10 e b'=2)

$$
\begin{aligned}
0.35 \cdot 2 &= 0 + 0.7 \\
0.7 \cdot 2 &= 1 + 0.4 \\
0.4 \cdot 2 &= 0 + 0.8 \\
0.8 \cdot 2 &= 1 + 0.6 \\
0.6 \cdot 2 &= 1 + 0.2 \\
0.2 \cdot 2 &= 0 + 0.4 \\
0.4 \cdot 2 &= 0 + 0.8 \\
&\ldots
\end{aligned}
$$

m

$$0.35_{10} = 0.\underbrace{010110\ldots}_{m}{}_{2}$$

(approssimazione $< 2^{-m}$)



- Per codificare i numeri frazionari di nuovo viene utilizzato il sistema di numerazione posizionale in base *2*

- Fissato il numero di bit *k* da utilizzare nella rappresentazione
  1. si converte il numero di partenza nella base *2*
  2. si considerano solo i primi *k* bit dopo la virgola, eventualmente tagliando i rimanenti o aggiungendo bit uguali a *0* alla parte finale fino ad ottenere complessivamente esattamente *k* bit

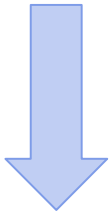
  NB: non è necessario rappresentare lo *0* iniziale e la virgola, perché sono comuni a tutti i numeri frazionari

Esempi *(k=4)*

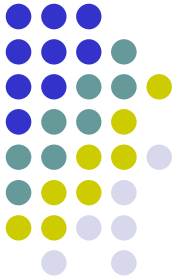
1) $0.375 = 0{,}011_2$

0110

2) $0.84375 = 0{,}11011_2$



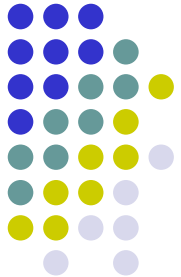
1101



Osservazioni:

- l'eventuale aggiunta di bit pari a *0* nel passo 2. è necessaria perché nella memorizzazione del numero bisogna specificare per ogni bit (anche per gli ultimi dopo la virgola che convenzionalmente non indichiamo quando pari a *0*) lo stato del relativo dispositivo bistabile; in caso contrario si potrebbero avere errori di rappresentazione
- l'eventuale taglio di bit finali (dopo il bit finale) causa un errore di rappresentazione; in particolare, se *n* è il numero frazionario da rappresentare e *r(n)* il numero rappresentato al posto di *n* tagliando dopo il bit *k*, l'errore assoluto commesso è

$$err.ass. = |n - r(n)| \leq 2^{-k}$$



Punti Chiave

- Codifica binaria dell'informazione
- Codifica caratteri
- Codifica numeri
  - Sistemi di numerazione posizionali
  - Conversioni di base
  - Codifica naturali