Sistemi di Numerazione Binaria
(a.a. 2008-2009)

Numeri e numerali

Numero: *entità astratta*

Numerale : *stringa di caratteri che rappresenta un numero in un dato sistema di numerazione*

- Lo stesso numero è rappresentato da numerali diversi in diversi sistemi

ESEMPIO

  – 156 nel sistema decimale
  – CLVI in numeri romani

- Il numero di caratteri del numerale determina l'intervallo di numeri rappresentabili

ESEMPIO

  – Interi a 3 cifre con segno in notazione decimale: [-999,+999]

Numeri a precisione finita

- In rappresentazioni *con numero finito di cifre* si perdono alcune proprietà:
  - chiusura degli operatori (+ , -, x)
  - proprietà associativa, distributiva,..

ESEMPIO

- 2 cifre decimali e segno [-99,+99]
- 78+36=114 (*chiusura della somma*)
- 60+(50-40) != (60+50)-40 (*proprietà associativa*)

- Si introducono errori di arrotondamento
- Si introducono buchi nella rappresentazione dei reali

ESEMPIO

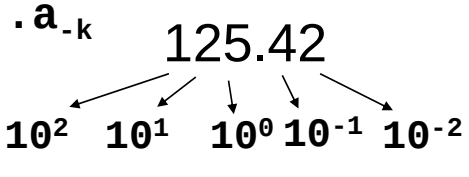
- Con numerali decimali con due sole cifre frazionarie non posso rappresentare correttamente 0.015

Sistemi posizionali

*Il numero è rappresentato come somma di potenze della base **b**, ciascuna moltiplicata per un coefficiente intero*

$$a_m\ a_{m-1}\ \ldots\ a_0\ .\ a_{-1}\ a_{-2}\ \ldots a_{-k}$$

$$N = \sum_{i=-k}^{m} a_i b^i \qquad 0 \le a_i \le b-1 \qquad b = \text{base}$$



- Ciascuna cifra del numerale rappresenta il coefficiente di una potenza della base
- L'esponente è dato dalla *posizione* della cifra

Aritmetica e notazione posizionale

- Lo scarso sviluppo dell'aritmetica in età classica è legato all'uso di una notazione non posizionale

  La notazione posizionale consente di effettuare le operazioni aritmetiche operando sui numerali

- Ad esempio, per sommare due numeri:
  - Si incolonnano i numerali
  - Si sommano cifre omologhe (coefficienti della stessa potenza della base)
  - Si propagano i riporti
  - Si ottiene il numerale che rappresenta la somma
- Invece nella notazione romana...

Simboli per le cifre

- Se la base è *b* occorrono *b* simboli per rappresentare le cifre del numerale
- Per la base 10 utilizziamo le cifre arabe (indiane!)
- Per basi inferiori a 10 se ne usa un sottoinsieme
- Per basi superiori a 10 occorre definire simboli aggiuntivi
- Casi di interesse:

b = 10 {0,9}
b = 2 {0,1}
b = 8 {0,1, ... 7}
b = 16 {0,1, ... 9,A,B,C,D,E,F}

Conversione decimale-binario

- Si effettuano divisioni ripetute per 2
- Il resto delle divisioni fornisce le cifre del numerale binario (a partire dalla meno significativa)

ESEMPIO $(26)_{10} = (11010)_2$

26	**0**	*cifra meno significativa*
13	**1**	
6	**0**	
3	**1**	
1	**1**	*cifra più significativa*
0		

Conversione decimale-binario (2)

- Altrimenti si può procedere 'ad occhio'
- i coefficienti della notazione binaria sono **0** o **1**
- Cioè un numero intero è rappresentabile come somma di un sottoinsieme delle potenze di 2
- Si cerca la più grande potenza di due contenuta nel numero, la si sottrae e si prosegue con la differenza
- Ogni cifra del numerale indica se la corrispondente potenza di due è presente nella somma o no

ESEMPIO $(26)_{10} = (11010)_2$

$$(26)_{10} = 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1$$

Intervalli rappresentati

- Rappresentando gli interi positivi e lo zero in notazione binaria con **n** cifre (bit) si copre l'intervallo **$[0,2^n-1]$**
- Si sfruttano tutte le **2^n** disposizioni

ESEMPIO **n=3** **$[0,2^3-1]$ -> [0,7]**

0	**000**
1	**001**
2	**010**
3	**011**
4	**100**
5	**101**
6	**110**
7	**111**

NB *Anche gli 0 non significativi devono essere rappresentati*

Ordini di grandezza binari

- In un sistema binario gli ordini di grandezza sono dati dalle potenze di 2

$2^0...2^9$ = 1,2,4,8,16,32,64,128,256,512..

2^{10} = 1024 **~ 10^3** **1 Kilobyte**

2^{20} = 2^{10} 2^{10} = 1048576 **~ 10^6** **1 Megabite**

2^{30} = 2^{10} 2^{10} 2^{10} = 1073741824 **~ 10^9** **1 Gigabyte**

2^{40} = ... = 1099511627770 **~ 10^{12}** **1 Terabyte**

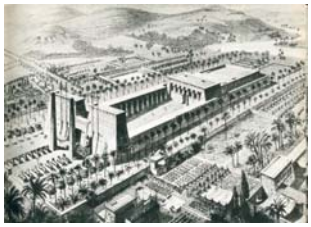
2^{50} = ... = 1125899906842624 **~ 10^{15}** **1 Petabyte**

Esempio

2^{26} = $2^6 \cdot 2^{20}$ = 64 M

1 Petabyte?

- How large is it?
- A quick comparison with one of the worst data loss in the story: the Alexandria library fire (270 a.d. ?)
- How many bytes were lost?
  - We are neglecting the quality...



Lost data

- Averaging data reported by historical writers we can assume about 50.000 lost books (~2^{16})
- Assuming (overestimation) each book size as Dante's Divina Commedia, 500.000 chars (~2^{19}), it results in

  $2^{16}*2^{19}=2^{35}$ = ~32 gigabytes

- 10% of my laptop hard disk...

The actual situation? A rough calculation

- The new Bibliotheca Alexandrina stores millions of books (2^{20})

- There are about 90.000 libraries in Europe and about 250.000 (~2^{18}) in the world:

$$2^{18}*2^{20}*2^{19}=2^{57}=2^{7}*2^{50}=\sim 128 \text{ petabytes}$$

- For the sake of simplicity and removing duplicates (how many copies of Divina Commedia are around?) we can conclude the calculation to

~1 petabyte of chars

1 petabyte

- **The entire written works of humankind, from the beginning of recorded history, in all languages...**

Interi positivi e negativi

- Per rappresentare gli interi relativi, a parità di cifre *si dimezza l'intervallo dei valori assoluti*
- Per esempio con **n = 3** bit possiamo rappresentare numeri compresi in valore assoluto tra **0** e **7**
- Si utilizzano varie rappresentazioni:
  - Modulo e segno
  - Complemento a **1**
  - Complemento a **2**
  - Eccesso $\mathbf{2^n}$

Rappresentazione in modulo e segno

- Analoga a quella che usiamo nella nostra notazione decimale
- Viene dedicato un bit per il segno e **n-1** bit per il modulo
- Convenzionalmente
  0 rappresenta il segno **+**
  1 rappresenta il segno **-**
- Intervallo di numeri rappresentati con n bit

$$[-2^{n-1}+1,\ +2^{n-1}-1]$$

ESEMPIO

n=4 bit intervallo [-7,+7]
5 = 0101 -5 = 1101

Intervallo simmetrico e doppia rappresentazione dello zero

Rappresentazione in complemento a 1

- Si aggiunge uno **0** a sinistra alla rappresentazione dei numeri positivi

Per cambiare di segno si complementa il numerale bit a bit

- I numerali positivi iniziano per **0**, i negativi per **1**
- Intervallo di numeri rappresentati con **n** bit:

$$[-2^{n-1}+1,\ +2^{n-1}-1]$$

ESEMPIO

n=4 bit intervallo [-7,+7]

5 = 0101 -5 = 1010

Intervallo simmetrico e doppia rappresentazione dello zero

Complemento a 1 (continua)

- La notazione CP1 è una *notazione posizionale*:

Pesi: $(-2^{n-1}+1)$ 2^{n-2} . . . 2^1 2^0

- Il bit di ordine più alto (a sinistra) ha come peso negativo pari ad una potenza di 2 meno uno: $(-2^{n-1}+1)$
- Per i numeri positivi il corrispondente coefficiente è **0**
- Per i numeri negativi il corrispondente coefficiente è **1:**
  - viene preso il *contributo negativo*
  - le altre cifre corrispondono a *contributi positivi*

ESEMPIO

-5 = 1010

-5 = 1 ·(-7) + 0 · 4 + 1·2 + 0 ·1

Rappresentazione in complemento a 2

- I numeri positivi hanno la stessa rappresentazione che in complemento a 1

I numerali negativi si ottengono sommando 1 alla loro rappresentazione in complemento a 1

- Intervallo di numeri rappresentati con n bit

$$[-2^{n-1},\ +2^{n-1}-1]$$

ESEMPIO

n=4 bit intervallo [-8,+7]

5 = 0101 -5 = 1011

Intervallo asimmetrico e semplice rappresentazione dello zero

Complemento a 2 (continua)

- Anche la notazione CP2 è una *notazione posizionale*:

Pesi: $(-2^{n-1})\ \ 2^{n-2}\ \ \ldots\ \ \ 2^{1}\ \ 2^{0}$

- Il bit di ordine più alto (a sinistra) ha come peso negativo pari ad una potenza di 2 : (-2^{n-1})
- Per i numeri negativi il primo coefficiente è **1** e gli altri corrispondono a *contributi positivi*

ESEMPIO

-5 = 1011

$$-5 = 1 \cdot (-8) + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1$$

- Regola pratica per complementare:

Partendo da destra si lasciano invariati tutti i bit fino al primo 1 compreso, e poi si complementa bit a bit

Rappresentazione in eccesso 2^{n-1}

- I numeri vengono rappresentati come somma fra il numero dato e una potenza di 2.
- Con **n** bit si rappresenta l'eccesso $\mathbf{2^{n-1}}$
- Intervallo come CP2: $\mathbf{[-2^{n-1}\ ,+2^{n-1}-1]}$
- Regola pratica:

I numerali in eccesso $\mathbf{2^{n-1}}$ *si ottengono da quelli in CP2 complementando il bit più significativo*

ESEMPIO

n=4 bit: eccesso **8**, intervallo **[-8,+7]**

- 3 **- 3+8=5** **: 0101**

+4 **+4+8=12** **: 1100**

Intervallo asimmetrico e semplice rappresentazione dello zero

Rappresentazioni in eccesso

- Possibili rappresentazioni in eccesso un numero qualsiasi **k**
- L'eccesso una potenza di **2** è solo un caso particolare, anche se molto interessante
- Rappresentando un intero **m** in eccesso **k** con **n** bit, si rappresenta in realtà il numero positivo **k+m**
- Deve comunque essere $\mathbf{k \leq 2^n}$
- L'intervallo rappresentabile dipende sia da **k** che da **n**:

$$\mathbf{[-k\ ,\ 2^n-k-1]}$$

ESEMPIO

n=8, k=127 **[-127,+128]**

n=8, k=100 **[-100,+155]**

n=8, k=50 **[-50,+205]**

Rappresentazioni a confronto

Decimale	M&S	CP1	CP2	Ecc 8
+ 7	0111	0111	0111	1111
+ 6	0110	0110	0110	1110
+ 5	0101	0101	0101	1101
+ 4	0100	0100	0100	1100
+ 3	0011	0011	0011	1011
+ 2	0010	0010	0010	1010
+ 1	0001	0001	0001	1001
+ 0	0000	0000	0000	1000
- 0	1000	1111	-----	-----
- 1	1001	1110	1111	0111
- 2	1010	1101	1110	0110
- 3	1011	1100	1101	0101
- 4	1100	1011	1100	0100
- 5	1101	1010	1011	0011
- 6	1110	1001	1010	0010
- 7	1111	1000	1001	0001
- 8	----	----	1000	0000

Notazione in base 16

- Per i numerali esadecimali occorrono 16 cifre

  { 0,1, ... 9,A,B,C,D,E,F }

- Conversione esadecimale-binario:

  Si fa corrispondere a ciascuna cifra esadecimale il gruppo di 4 bit che ne rappresenta il valore

ESEMPIO

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

- Conversione binario-esadecimale:

  Partendo da destra si sostituisce a ciascun gruppo di 4 o meno cifre binarie la cifra esadecimale che ne rappresenta il valore

Numerali e numeri

- Un *numerale* è solo una stringa di cifre
- Un numerale rappresenta un numero solo se si specifica un sistema di numerazione
- Lo stesso numerale rappresenta diversi numeri in diverse notazioni

ESEMPIO

La stringa **110100** rappresenta:

- *Centodiecimilacento* in base 10
- $\mathbf{(+52)_{10}}$ in binario naturale
- $\mathbf{(-11)_{10}}$ in complemento a 1
- $\mathbf{(-12)_{10}}$ in complemento a 2
- $\mathbf{(+20)_{10}}$ in eccesso 32
- In esadecimale un numero dell'ordine di vari milioni

Addizioni binarie

- Le addizioni fra numerali binari si effettuano cifra a cifra (come in decimale) portando il riporto alla cifra successiva

```
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0   con il riporto di 1
```

ESEMPIO: `3 + 2 = 5`

```
  0011 +
  0010 =
  ----
  0101
```

Se il numero di cifre non permette di rappresentare il risultato si ha un trabocco nella propagazione del riporto

Addizioni in complemento

- In CP2 somme e sottrazioni tra numerali sono gestite nello stesso modo, *ma si deve ignorare il trabocco*:

```
 4 +      0100 +
 2 =      0010 =
 6        0110
```

- Se gli operandi hanno segno diverso il risultato è sempre corretto:

```
 4 +      0100 +
-1 =      1111 =
 3       10011
```

- Se i due operandi hanno lo stesso segno e il risultato segno diverso c'è errore

```
 6 +      0110 +
 3 =      0011 =        ( 9 non cade nell'intervallo )
 9        1001
```


Conversioni in eccesso 2^{n-1}

- Dato un numero **m d**eterminare il numero minimo di cifre n_{min} necessarie
- Determinare la prima potenza di **2** superiore al modulo di *m* e confrontarla con gli estremi dell'intervallo

ESEMPIO: convertire $(-347)_{10}$ in eccesso 2^{n-1}

$$2^8 = 256 < 347 < 512 = 2^9$$

intervallo con **n** *bit*: $[-2^{n-1}, +2^{n-1}-1]$ *pertanto* $n_{min}=10$

$$512-347 = 165$$

$$165 = 128+32+4+1$$

$(-347)_{10}$ in eccesso 2^9 *è:*

512	256	128	64	32	16	8	4	2	1
0	0	1	0	1	0	0	1	0	1

Conversioni in CP1 e CP2

- Se il numero è *negativo*:
  - **a)** determinare il numero di bit **n**
  - **b)** convertire il positivo corrispondente in notazione a n bit
  - **c)** complementare il numerale così ottenuto (a 1 o a 2)

***ESEMPIO**:* convertire $\mathbf{(-347)_{10}}$ in CP2

$\mathbf{2^8 = 256 < 347 < 512 = 2^9}$ **->** n_{min}**=10**

$\mathbf{(-347)_{10}}$ *in notazione a 10 bit è:*

512	256	128	64	32	16	8	4	2	1
0	1	0	1	0	1	1	0	1	1

quindi, complementando a 2

− 512	256	128	64	32	16	8	4	2	1
1	0	1	0	1	0	0	1	0	1

Rappresentazione di numeri reali

- Con un numero finito di cifre è solo possibile rappresentare un numero razionale *che approssima con un certo errore* il numero reale dato
- Vengono usate due notazioni:

A) <u>Notazione in virgola fissa</u>

Dedica parte delle cifre alla parte intera e le altre alla parte frazionaria

± XXX .YY

B) <u>Notazione in virgola mobile</u>

Dedica alcune cifre a rappresentare un esponente della base che indica l'ordine di grandezza del numero rappresentato

Notazione in virgola mobile

- Estende l'intervallo di numeri rappresentati a parità di cifre, rispetto alla notazione in *virgola fissa*
- Numeri reali rappresentati da una coppia di numeri **<m,e>**

m : mantissa normalizzata tra due potenze successive della base

$$b^{i-1} \leq |\ m\ | < b^{i}$$

– **e** : esponente intero con segno

$$n = m \cdot b^{e}$$

- Sia **m** che **e** hanno un numero prefissato di cifre

Intervalli limitati ed errori di arrotondamento

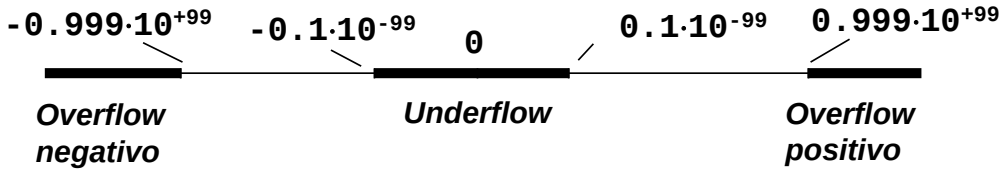
Esempio in base 10

- Numerali a **5** cifre ± **.XXX** ± **EE**
- *Mantissa* : **3** cifre con segno

$$0.1 \leq |m| < 1$$

- *Esponente*: 2 cifre con segno

$$-99 \leq e \leq +99$$

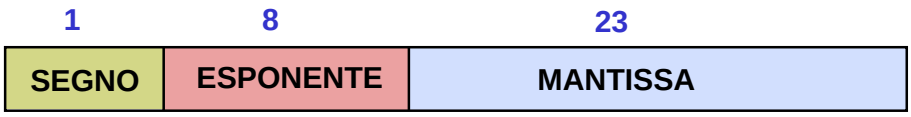


Con le stesse **5** cifre in notazione a punto fisso ± **XXX .YY** :

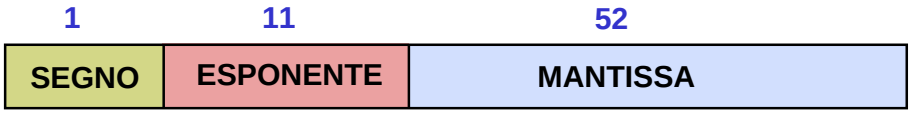
- L'intervallo scende **[-999.99,+999.99]**
- Ma si hanno **5** cifre significative invece di **3**

Standard IEEE 754 (1985)

- Formato non proprietario cioè indipendente dall'architettura
- Semplice precisione a 32 bit:

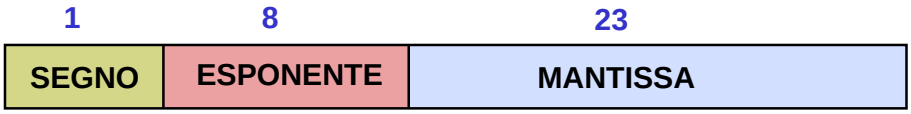


- Doppia precisione a 64 bit



- Notazioni in modulo e segno
- Alcune configurazioni dell'esponente sono riservate

IEEE 754 a 32 bit



- **ESPONENTE**
  - Rappresentato in *eccesso 127*
  - L'intervallo è **[-127, +128]**
  - Le due configurazioni estreme non si usano, quindi:

$$-126 \leq e \leq 127$$

- **MANTISSA**
  - È sempre *normalizzata*
  - Se ne rappresenta *solo la parte frazionaria*

Due rappresentazioni, *a seconda del valore dell'esponente*:

A) Numeri normalizzati $e \neq 00000000$

B) Numeri denormalizzati $e = 00000000$

Numeri normalizzati

- Un numerale si intende in questa rappresentazione quando:

  $e \neq 00000000$, $e \neq 11111111$

- La mantissa è normalizzata tra **1** e **2**:

  $1 \leq m < 2$

- Quindi è sempre nella forma:

  1.XXXXXXXXX…

- I **23** bit rappresentano la sola parte frazionaria
- Gli intervalli di numeri rappresentati sono pertanto:

  $(-2^{128}, -2^{-126}]$ $[2^{-126}, 2^{128})$

- Gli estremi sono esclusi perché il massimo valore assoluto di **m** è molto vicino a 2 ma comunque inferiore
- L'intervallo $(-2^{-126}, 2^{-126})$ è *intervallo di underflow*

Numeri denormalizzati

- Un numerale si intende in questa rappresentazione quando:

  $e = 00000000$

- L'esponente assume il valore *convenzionale* **e=-126**
- La mantissa è normalizzata tra **0** e **1**:

  $0 < m < 1$

- Quindi è sempre nella forma:

  0.XXXXXXXXX…

- I **23** bit rappresentano la sola parte frazionaria
- La più piccola mantissa vale 2^{-23}
- Gli intervalli rappresentati sono:

  $(-2^{-126}, -2^{-149}]$ $[2^{-149}, 2^{-126})$

Più piccola è la mantissa minore è il numero di cifre significative

Altre configurazioni

- Lo Standard IEEE 754 attribuisce valori convenzionali a particolari configurazioni di **e** ed **m**
  - **A)** **e** ed **m** tutti **0** rappresentano il valore **0** (*altrimenti non rappresentabile*)
  - **B)** **m** tutti **0** ed **e** tutti **1** rappresentano l'*overflow*
  - **C)** **m** ≠ **0** ed **e** tutti **1** indicano la situazione *Not A Number (NAN)*, cioè un valore indefinito (ad es. il risultato di una divisione per **0**)

Queste convezioni sono una caratteristica peculiare della notazione IEEE 754; non valgono, se non esplicitamente definite, per altre notazioni (e.g., compiti di esame)

IEEE 754: estremi degli intervalli

- *Più grande normalizzato* $\sim 2^{128}$:

  X 11111110 11111111111111111111111

  ± 2^{127} ~ 2

- *Più piccolo normalizzato* 2^{-126} :

  X 00000001 00000000000000000000000

  ± 2^{-126} 1

- *Più grande denormalizzato* $\sim 2^{-126}$:

  X 00000000 11111111111111111111111

  ± 2^{-126} $(0.11\ldots)_2 \sim 1$

- *Più piccolo denormalizzato* 2^{-149} :

  X 00000000 00000000000000000000001

  ± 2^{-126} $(0.00\ldots1)_2 = 2^{-23}$

Addizioni in virgola mobile

Per addizionare e sottrarre occorre portare i numeri allo stesso esponente e scalare le mantisse

ESEMPIO: $n_1 + n_2$ *in notazione IEEE 754*

n_1 : 0 10011001 00010111011100101100111
n_2 : 0 10101010 11001100111000111000100

$e_1 = (26)_{10}$, $e_2 = (43)_{10}$:
occorre scalare m_1 *di 17 posti*

n'_1 : 0 10101010 00000000000000001000101 +
n_2 : 0 10101010 11001100111000111000100 =
0 10101010 11001100111001000001001

Notare che l'addendo più piccolo perde cifre significative

Moltiplicazioni fra interi

- L'operazione viene effettuata sui numerali, come in decimale
- Il numerale che si ottiene rappresenta il risultato
- La tabellina delle moltiplicazioni però è molto più semplice che nel caso decimale **2 x 2** invece di **10 x 10**

	0	1
0	0	0
1	0	1

Come in decimale, si incolonnano e si sommano i prodotti parziali scalandoli opportunamente

Moltiplicazioni fra interi: esempio

5 · 11 = 55

$(11)_{10}$	1011	x
$(5)_{10}$	101	=
	1011	
	0000	
	1011	
$(55)_{10}$	110111	

Notare che, in base alla tabellina, ciascun prodotto parziale è pari a zero oppure al moltiplicando

Moltiplicazioni in virgola fissa

Si opera come in decimale, tenendo conto del numero di cifre frazionarie e riposizionando il punto frazionario

2.75 1.25 = 3.4375

$(2.75)_{10}$	10.11	x
$(1.25)_{10}$	1.01	=
	1011	
	0000	
	1011	
$(3.4375)_{10}$	11.0111	

Moltiplicare o dividere per 2^n equivale a spostare il punto di n *posti a destra o a sinistra*

Moltiplicazioni in virgola mobile

Si moltiplicano le mantisse e si sommano algebricamente gli esponenti e, se necessario, si scala la mantissa per normalizzarla e si riaggiusta l'esponente

ESEMPIO: $n_3 = n_1 \times n_2$

n_1 : 0 10011001 10010111011100101100111
n_2 : 1 10101010 10000000000000000000000

$e_1 = (26)_{10}$, $e_2 = (43)_{10}$

$e_1 + e_2 = (69)_{10} = 11000100$
$m_1 \times m_2 = 10.011000110010101110110101$

si scala la mantissa di un posto
si aumenta di 1 l'esponente

n_3 : 1 11000101 00110001100101011101101

Errore assoluto e relativo

- Rappresentando un numero reale **n** in una notazione floating point si commette un errore di approssimazione
- In realtà viene rappresentato un numero razionale **n'** con un numero limitato di cifre significative

ERRORE ASSOLUTO: $e_A = n - n'$

ERRORE RELATIVO: $e_R = e_A / n = (n - n')/n$

- L'ordine di grandezza dell'**errore assoluto** dipende dal *numero di cifre significative* e dall'*ordine di grandezza del numero*
- L'ordine di grandezza dell'**errore relativo** dipende solo dal *numero di cifre significative*

Errore relativo massimo

Se la mantissa è normalizzata l'errore relativo è sempre inferiore ad un'unità sull'ultima cifra rappresenta

- Con **k** cifre frazionarie e mantissa normalizzata tra **1** e **2** si hanno sempre **k + 1** cifre significative
- L'errore relativo massimo è ordine $\mathbf{2^{-k}}$

ESEMPIO

Con 10 cifre frazionarie l'errore massimo è ordine $\mathbf{2^{-10}}$

Su un numero di ordine $\mathbf{2^{m}}$ *l'errore assoluto è dato da* $\mathbf{e_A = e_R \cdot n}$ *e quindi è ordine* $\mathbf{2^{m-10}}$

Nelle notazioni non normalizzate il numero di cifre significative, e quindi l'errore relativo massimo non è costante

Errori numerici

Errori di calcolo:

Nel 1996 l' ESA (Agenzia Spaziale Europea) lancia il missile Arianna 5 (10 anni di lavoro, 7 bilioni di $) con un carico di alcune tonnellate di satelliti. Dopo 39 secondi di volo, a circa 1km di altezza, il sistema di autodistruzione entra in azione e distrugge il missile ed i preziosi satelliti a bordo. Perche?

- Il sistema di assetto inerziale (giroscopi, accelerometri, ecc.) era gestito da un doppio computer (hw ridondante) e, tra le altre cose, monitorava la velocità orizzontale che, una volta acquisita, veniva passata al sitema di guida subendo una **conversione da 64 a 16 bit**;
- Dopo circa 36 secondi la velocità orizzontale supera il massimo memorizzabile in 16 bit, valore alto, ma ragionevole per un missile così veloce, e nella conversione si verifica un **overflow**;
- Il controllo passa immediatamente al computer ridondante che interviene istantaneamente e rieffettua i calcoli. Ma il SW è lo **stesso** (!!!) e si verificalo **stesso overflow**;
- Il secondo computer restituisce un **codice di errore** ed effettua uno shutdown (36.7 secondi dopo il lancio)
- Il **codice di errore** viene **interpretato** dal sistema di guida come un **assetto di volo** estremamente bizzarro ed il sistema di guida reagisce immediatamente, imponendo al missile una brusca virata
- A causa dell'accelerazione il motore principale rischia di staccarsi dal missile
- Il sistema di autodistruzione interviene prontamente...