Università degli Studi di Milano
CdL in Informatica per la comunicazione digitale
AA 2025/2026

Architettura degli elaboratori
Lez 3:

Rappresentazione di numeri frazionari – Parte A

Marco Tarini
marco.tarini@unimi.it

1

Nota linguistica preliminare sul punto e la virgola

- In Italiano, per dire «mille euro e cinquantacinque centesimi», scriviamo:

1.000,55 €

spesso omesso

- In Inglese, scriviamo invece la stessa cifra come

1,000.55 €

spesso omessa

- In questo corso, useremo sempre la notazione Inglese per i numeri (perché è standard in informatica, per es, nei linguaggi di programmaz.)

- 2 -

2

Numeri binari non interi

- Come si esprime un numero frazionario in base 10?

10^2 10^1 10^0 10^{-1} 10^{-2}

235.13_{10}

=

$2x10^2 + 3x10^1 + 5x10^0 + 1x10^{-1} + 3x10^{-2}$

2 centinaia + 3 decine + 5 unità + 1 decimi + 3 centesimi .

- 3 -

3

Numeri binari non interi

- Come si esprime un numero non intero in base 2 (binario)?
  Stesso principio

2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3} 2^{-4}

101.1001_{10}

=

$1x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1} + 0x2^{-2} + 0x2^{-3} + 1x2^{-4}$

$= 1x4 + 0x2 + 1x1 + 1x½ + 0x¼ + 0x1/8 + 1x1/16$

$= 4 + 1 + 0.5 + 0.0625$

$= 5.5625_{10}$

- 4 -

4

Numeri binari non interi

- oppure: cosi come, in base 10:

235.13	235.138
=	=
235 unità + 13 centesimi ($/10^2$)	235 unità + 138 millesimi ($/10^3$)
(perché sono 2 cifre dopo la virgola)	(perché sono 3 cifre dopo la virgola)

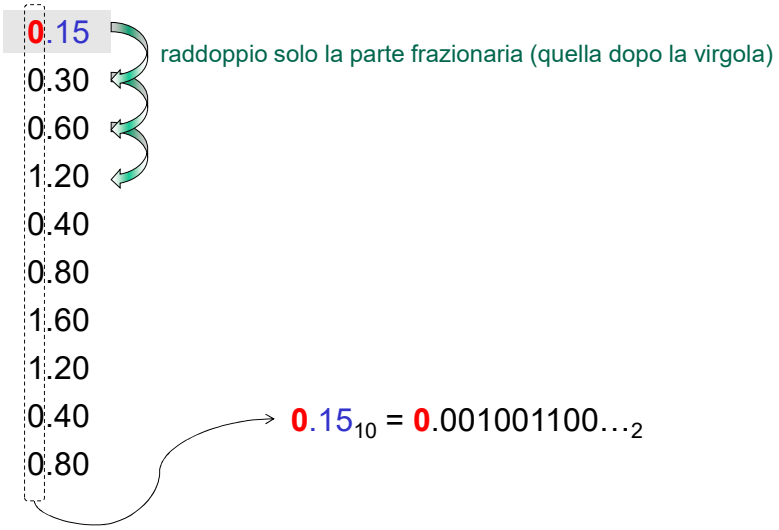
- nello stesso modo, in base 2:

$$101.1001_2$$
$$=$$
$$101_2 \text{ unità} + 1001_2 / 2^4$$
$$= 5 + 9/16$$

- 5 -

5

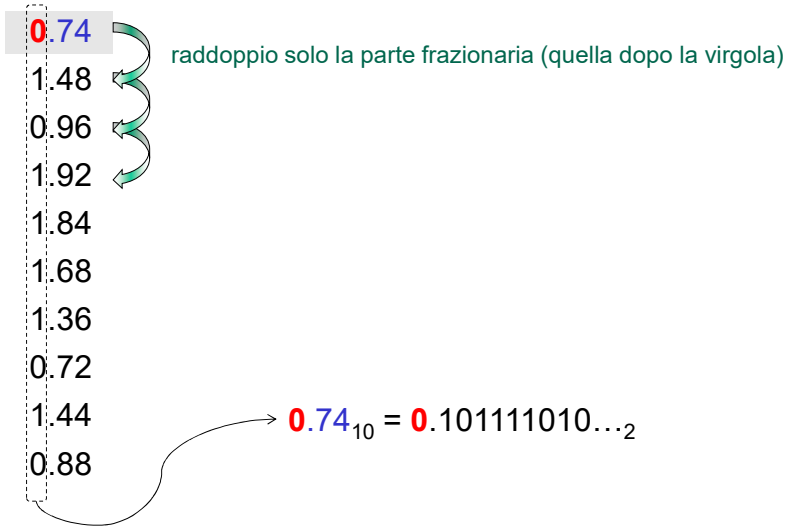
Conversione da decimale a binario di numeri frazionari: esempio



- 6 -

6

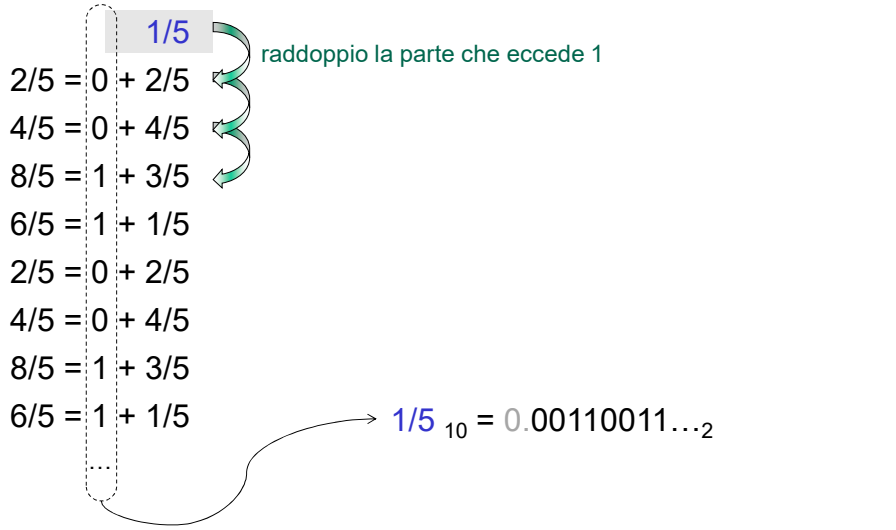
Conversione da decimale a binario di numeri frazionari: esempio



- 7 -

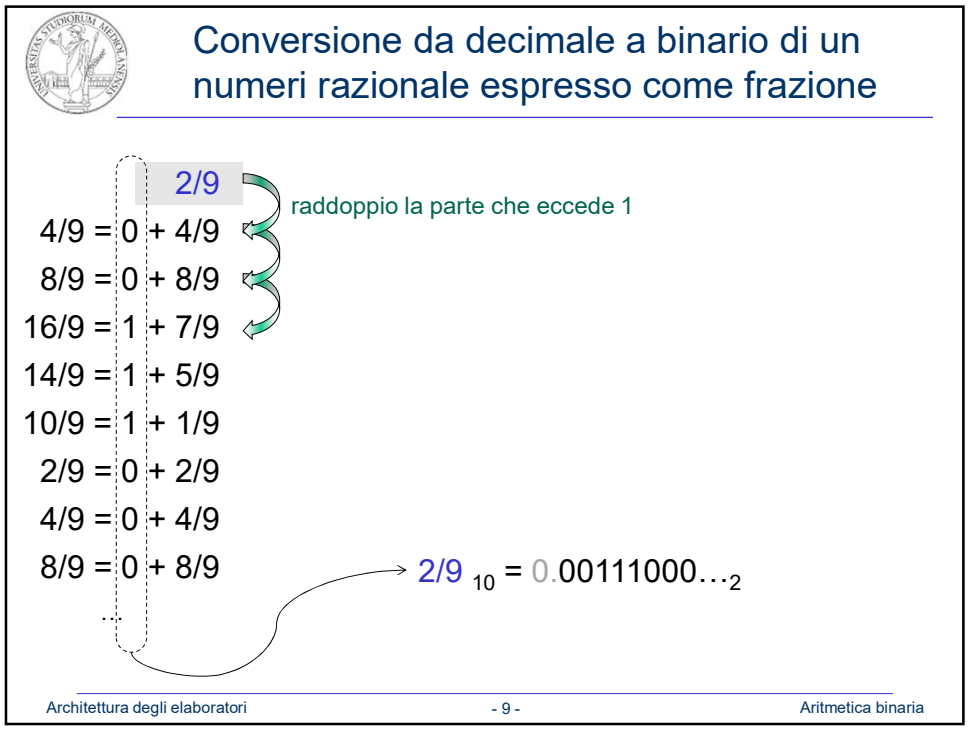
7

Conversione da decimale a binario di un numeri razionale espresso come frazione

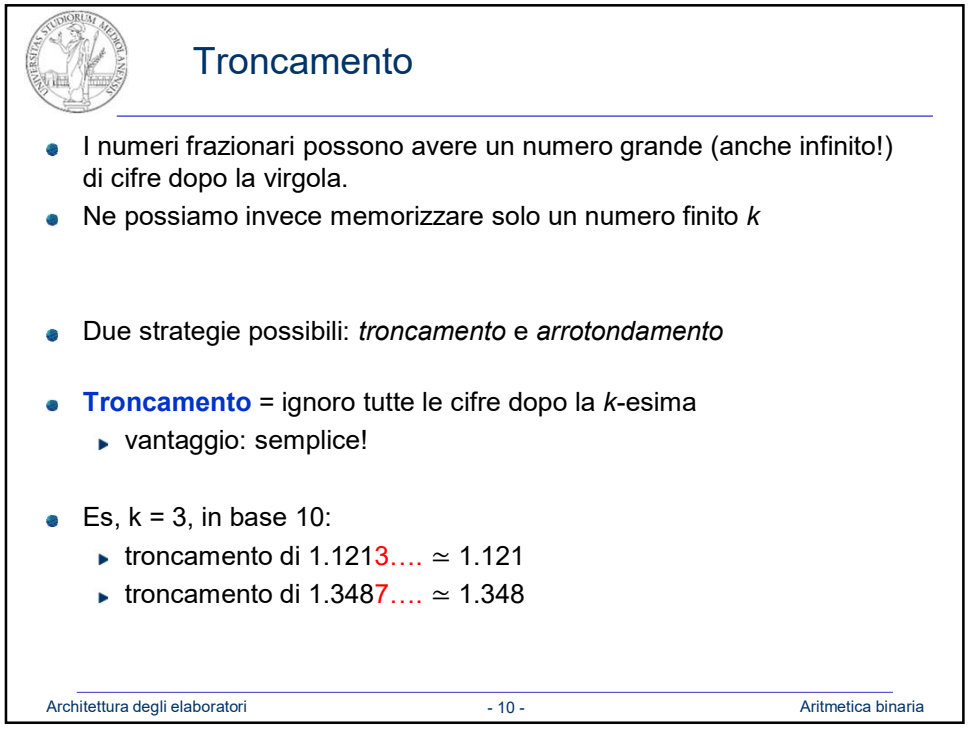


Architettura degli elaboratori - 8 - Aritmetica binaria

8


Conversione da decimale a binario di un numeri razionale espresso come frazione
2/9
raddoppio la parte che eccede 1
4/9 = 0 + 4/9
8/9 = 0 + 8/9
16/9 = 1 + 7/9
14/9 = 1 + 5/9
10/9 = 1 + 1/9
2/9 = 0 + 2/9
4/9 = 0 + 4/9
8/9 = 0 + 8/9
...
2/9 10 = 0.00111000...2
Architettura degli elaboratori
- 9 -
Aritmetica binaria

9


Troncamento
I numeri frazionari possono avere un numero grande (anche infinito!) di cifre dopo la virgola.
Ne possiamo invece memorizzare solo un numero finito k
Due strategie possibili: troncamento e arrotondamento
Troncamento = ignoro tutte le cifre dopo la k-esima
vantaggio: semplice!
Es, k = 3, in base 10:
troncamento di 1.12133.... ≃ 1.121
troncamento di 1.34877.... ≃ 1.348
Architettura degli elaboratori
- 10 -
Aritmetica binaria

10

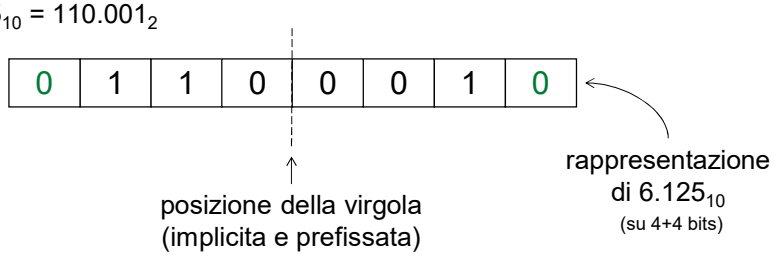
Arrotondamento

- **Arrotondamento** = scarto tutte le cifre dopo la *k*-esima, ma prima scelgo se approssimare per eccesso o per difetto,
  - scopo: minimizzare l'errore introdotto dall'approssiamzione
  - strategia: dipende solo della prima cifra che scarto
  - in base 10: se è 0,1,2,3,4 → per difetto.
    se è 5,6,7,8,9 → per eccesso
  - in base 2: se è 0 → per difetto
    se è 1 → per eccesso
- Es, con *k* = 3, in base 10:
  - 1.121**3**26…. ≃ 1.121 (per difetto: scarto e basta)
  - 1.348**7**61…. ≃ 1.349 (per eccesso: aggiungo 0.001 e scarto)
- Es, con *k* = 3, in base 2:
  - 0.101**0**10… ≃ 0.101 (per difetto)
  - 0.101**1**01… ≃ 0.101 + 0.001 = 0.110 (per eccesso)

11

Rappresentazione di numeri frazionari «in virgola fissa»

- Idea: per rappresentare un numero frazionale:
  - converto in base 2
  - memorizzo la parte intera in *k* bits, la parte dopo la virgola in *h* bit
  - (totale bits usati = *k* + *h*)
  - arrotondo a *h* cifre la parte decimale
  - con *h* e *k* decisi una volta per tutte
- Esempio. Su un byte (8 bit), *k* = 4 e *h* = 4:
  $6.125_{10} = 110.001_2$



12

Rappresentazione di numeri frazionari «in virgola fissa»: proprietá

- nota: arrotondo = commetto un errore
  (precisione numerica limitata)
- Limiti:
  - Max numero esprimibile: 2^k - 1 + (quasi) 1 = (quasi) 2^k
  - Min numero esprimibile > 0 : 2^{-h} (← la «precisione»!)
    (i numeri più piccoli sono arrotondati a 0 !)
- Questi limiti sono molto stretti.
- Vorremmo poter rappresentare numeri sia MOLTO più grandi che MOLTO più vicini a zero.
- Vediamo una rappresentazione più potente ed espressiva…

La notazione scientifica (in base 10)

- In molte occasioni (soprattutto in contesto scientifico) vogliamo rappresentare numeri sia molto grandi che molto vicini allo 0
- per es:
  - dalla massa dell'elettrone
    0.00000000000000000000000000000913 grammi
  - alla massa del sole:
    circa 1,900,000,000,000,000,000,000,000,000,000,000 grammi
- Nota: anche se questi numeri cadono in un intervallo di circa 60 ordini di grandezza (sessanta «zeri» di differenza),
  nessuno in pratica usa mai numeri di 60 cifre.
- Infatti, la precisione richiesta è sempre minore, ad es:
  - a nessuno interessa quale sia la cifra esatta corrispondente ai grammi (o alle tonnellate), quando si esprime la massa del sole
  - o il numero esatto di micron (0.000001m) quando si esprime l'altezza di una persona

La notazione scientifica

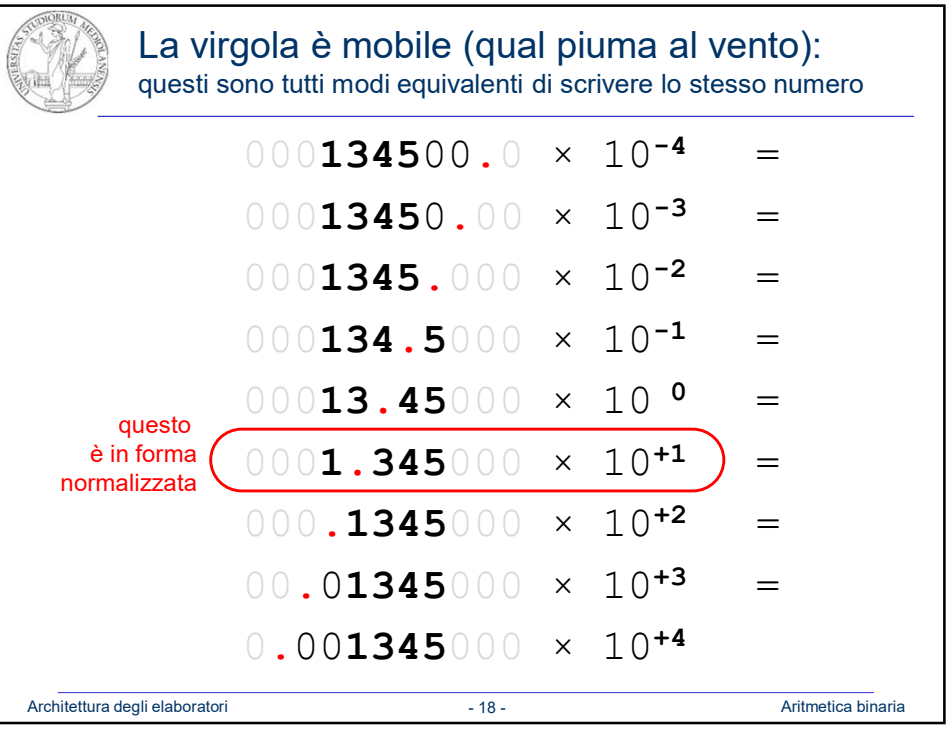
- Ne consegue che di solito si adotta la **notazione scientifica**, sia per rappresentare questi valori (e anche per far conti su di essi)
- Per esempio:
  - non 0.0000000000000000000000000000913
    ma 9.13×10^{-28}
  - non 1,900,000,000,000,000,000,000,000,000,000,000
    ma 1.9×10^{33}
- Nota: questo è utile anche per i numeri *interi* ma grandi
  - Es: numero di abitanti sulla terra:
    circa 8,120,000,000 = 8.12×10^{9}
    (magari non sappiamo neanche la cifra dell'unità, o delle decine)
- Scorciatoia sintattica :
  - Il numero 1.9×10^{33} si può scrivere anche come: 1.9e33
  - Il numero 9.13×10^{-28} si può scrivere anche come: 9.13e-28
  - Molto utile per scriverli con una tastiera!

Notazione scientifica (in base 10)

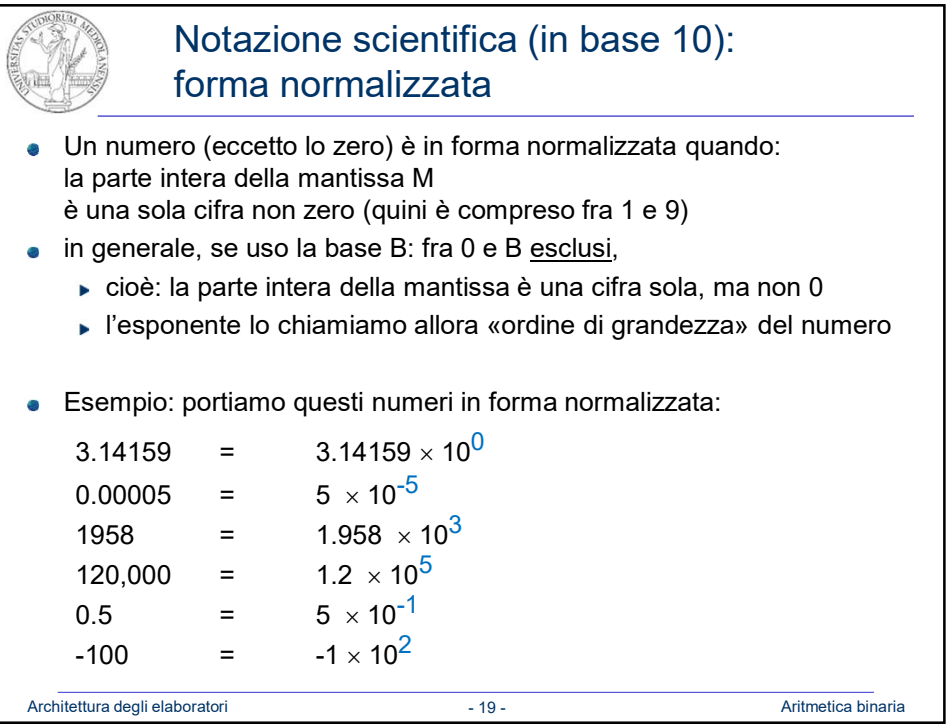
$$\pm m \times 10^{n}$$

Usatissima in tutte le discipline tecniche e scientifiche:

- m: *mantissa* (un numero con la virgola)
- n: *esponente* (un numero intero, negativo o positivo)
- Consente di manipolare agevolmente grandezze anche molto diverse (nella stessa unità di misura!)
- Ad esempio, per le lunghezze:
  dallo spessore di un capello: 78.0×10^{-6} metri
  al diametro del sole: 1.39×10^{9} metri


La virgola è mobile (qual piuma al vento):
questi sono tutti modi equivalenti di scrivere lo stesso numero
000134500.0 × 10-4 =
00013450.00 × 10-3 =
0001345.000 × 10-2 =
000134.5000 × 10-1 =
000013.4500 × 10 0 =
questo è in forma normalizzata
0001.345000 × 10+1 =
000.1345000 × 10+2 =
00.01345000 × 10+3 =
0.001345000 × 10+4
Architettura degli elaboratori
- 18 -
Aritmetica binaria

18


Notazione scientifica (in base 10):
forma normalizzata
Un numero (eccetto lo zero) è in forma normalizzata quando:
la parte intera della mantissa M
è una sola cifra non zero (quini è compreso fra 1 e 9)
in generale, se uso la base B: fra 0 e B esclusi,
cioè: la parte intera della mantissa è una cifra sola, ma non 0
l'esponente lo chiamiamo allora «ordine di grandezza» del numero
Esempio: portiamo questi numeri in forma normalizzata:
3.14159 = 3.14159 × 10^0
0.00005 = 5 × 10^-5
1958 = 1.958 × 10^3
120,000 = 1.2 × 10^5
0.5 = 5 × 10^-1
-100 = -1 × 10^2
Architettura degli elaboratori
- 19 -
Aritmetica binaria

19

Notazione scientifica in base 2

- valore = $\pm M \times 2^E$
  - M: «mantissa», un numero con la virgola in base 2.
  - E: esponente: un numero intero (con segno)
- Esempio: $+10.1011_2 \times 2^5 = +1010110_2 = 64+16+4+2 = 86$
- Esempio: $+110.01_2 \times 2^{-4} = +0.011001_2 = 11001_2 / 2^6 = 25 / 64$
- Il numero è «normalizzato» se la parte intera di M è una sola cifra non zero (quindi, se è uno!)

- 20 -

20

Notazione scientifica normalizzata in base 2

- valore = $\pm 1.M \times 2^E$
  - M: parte frazionaria della «mantissa», (la parte dopo la virgola). O anche solo la «mantissa».
  - E: esponente: un numero intero (con segno)
- Esempi: $-1.011_2 \times 2^6 = -1011000_2 = 88$
  $+1.000_2 \times 2^{-2} = +0.01_2 = ¼$

- 21 -

21

Come si memorizzano davvero i numeri frazionari in un elaboratore

- Idea: rappresentare in una quantità fisso di bits un numero nella la sua versione **scientifica normalizzata in base 2**,

  $\pm 1.M \times 2^E$

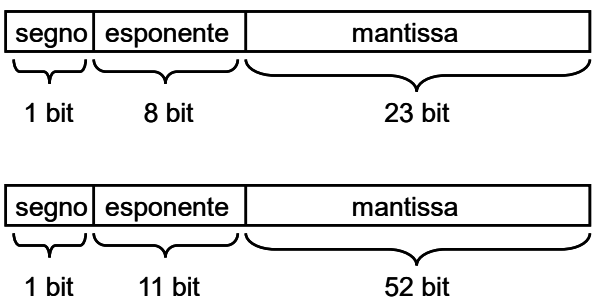
- attraverso una sequenza di bit:
  - Segno ±, con un bit
  - Mantissa M, con *k* bits
    (ma solo la parte dopo la virgola, senza bisogno di riportare la cifra delle unità, che tanto è sempre 1)
  - Esponente E, con *h* bits

- Dobbiamo solo decidere (una volta per tutte!) convenzioni come:
  - Come codificare il segno (per es: 0 se positivo, 1 se negativo)
  - Quanti bit di mantissa *k*, e di esponente *h*?
  - Come codificare l'esponente, che è un numero *con segno*? (per es: complemento a due, forse?)

22

Lo standard IEEE 754

- È lo standard universalmente adottato per la rappresentazione di numeri in virgola mobile (floating point, anche detti «float»)
- Definisce alcuni formati, fra cui i più usati:
  - Singola precisione (32 bit) --- spesso detti semplicemente `float`
  - Doppia precisione (64 bit) --- speso detti `double`



23

Lo standard IEEE 754: scelte decise una volta per tutte

- Per il formato a 32 bit (precisione singola):
  - 1 bit di segno (0 positivo, 1 negativo)
  - 8 bit di esponente,
    con offset a 127 (nota: non CP2).
    Per esempio, se codifico $0010\ 1010_2 = 42_{10}$,
    l'esponente E vale 42 - 127 = -85
  - 23 bit di mantissa (solo la parte dopo la virgola)
    Per esempio, se codifico 0101 0010 … etc
    la mantissa vale **1.**0101 0010 … etc
- Per il formato a 64 bit (precisione doppia):
  - Segno: come sopra
  - 11 bit di esponente, con offset a 1023 (nota: non CP2).
  - 52 bit di mantissa (solo la parte dopo la virgola, come sopra)

24

IEEE-754 per float a 32 bit: note sulla rappresentazione dell'esponente

Codifica **offset 127** (anche detta 'spiazzamento 127'):
Cioè si somma all'esponente da rappresentare l'offset 127.
Esempi:

esponente -1 viene rappresentato come +126 (0111 1110)
esponente -126 viene rappresentato come +1 (0000 0001)

Sono quindi rappresentabili gli esponenti da -127 a +128. Compresi? (no, il realtà, *esclusi*, perché il massimo e minimo esponente, -127 e +128, corrispondono configurazioni «speciali». Vedi prossima lezione)

Per esponenti <=0 il bit più significativo viene sempre 0
(es: esponente 0 → 0111 1111)
Per esponenti >0 il bit più significativo viene sempre 1
(es: esponente +1 → 1000 0000)

Questo vale in tutti formati floating point IEEE, perché in ciascuno di loro l'offset vale 2^(k-1) -1, con k il numero di bit dell'esponente.

25

Esempio: rappresentare –10.75 in float
(virgola mobile a 32 bit, standard IEEE-754)

- Passi:
  - Conversione a binario: $-10.75_{10} = -1010.11_2$
  - Normalizzazione: -1.01011×2^3
  - Codifica del segno: "–" = 1
  - Codifica dell'esponente:

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

10.75_{10} =

___8___ _____ 23 _____

= 1 | 1000 0010 | 01011000000000000000000

26

Esercizio: lo standard «half-precision»

- Codifichiamo il numero -33.25 in nel formato float a 16 bit (2 byte), detto «half-precision» (e meno usato) che ha:
  - un bit di segno, come al solito
  - solo 5 bit di esponente, con offset a 15
  - solo 10 bit di mantissa
- Quali valgono questi questi due byte, espressi in esadecimale?
- Passo 1: convertiamo in base 2: $-33.25_{DEC} = -10001.01_{BIN}$
- Passo 2: forma normalizzata: -1.000101×2^4
- Quindi:
  - Segno: 1 (negativo)
  - Mantissa (senza lo «1.» iniziale): 0001010000
  - Esponente: «4_{DEC}» codificato con $4+15 = 19_{DEC}$ cioè 10011
- Totale: 1 10011 0001010000 (cioè: 1100 1100 0101 0000)
- In esadecimale: **0xCC90**, e i due byte sono **0xCC** e **0x90**

Architettura degli elaboratori - 27 -

27

Esercizi di riepilogo

- Converti in base 10 i numeri binari 1110.1101 e 0.0111
- Somma i due numeri sopra *in binario*, e converti in base 10 il risultato (verifica che venga lo stesso risultato ottenuto sommando i due numeri in decimale)
- Converti in binario i numeri decimali 17.265 e 0.125
- Scrivi come numeri con la virgola, in binario: 5/9, 1/6
- Rappresenta in virgola fissa su un byte, con 5 bit per la parte intera e 3 per la parte frazionaria, i numeri di cui sopra, *con arrotondamento*
- Esprimere in forma normalizzata (binaria) gli stessi quattro numeri
- Rappresenta i numeri di cui sopra in modo conforme allo standard IEEE 754 su 16 bit (cioè due byte).
  Esprimi questi byte in forma esadecimale (come per es «0xA1»)
- Scopri quali byte (e quanti) costituiscono la rappresentazione di «1» in *double precision* (secondo lo standard IEEE 754)

28

Per un approfondimento

- Testo: Struttura e progetto dei calcolatori,
  Patterson & Hennessy Cap. 3.5
- Esercizi del Capitolo 3.13
  (quelli rilevanti)

29