Rappresentazione in virgola mobile

Architetture dei Calcolatori
(lettere A-I)

Rappresentazione di numeri reali

- Con un numero finito di cifre è possibile rappresentare solo un numero razionale che *approssima* con un certo *errore* il numero reale dato
- Vengono usate due notazioni
  - Notazione in virgola fissa
    - Dedica parte delle cifre alla parte intera e le altre alla parte frazionaria (la posizione della virgola è fissata su un bit prestabilito): ± XXX .YY
  - Notazione in virgola mobile
    - Dedica alcune cifre a rappresentare un esponente della base che indica l'ordine di grandezza del numero rappresentato

Perché la rappresentazione in virgola mobile

- Limitazioni della rappresentazione in virgola fissa
  - Non rappresenta bene numeri frazionari molto grandi
  - Non rappresenta bene numeri (frazioni) molto piccoli
- La rappresentazione in virgola mobile estende l'intervallo di numeri rappresentati a parità di cifre, rispetto alla notazione in *virgola fissa*
- Si può usare la *notazione scientifica*
  - Si esprime 432 000 000 000 come 4.32×10^{11}
  - Le 11 posizioni dopo il 4 vengono espresse dall'esponente
- Principio della rappresentazione in virgola mobile (detta anche *floating point*)
  - Si fa scorrere la virgola decimale fino ad una posizione conveniente, tenendo conto di ogni spostamento con l'esponente

Rappresentazione in virgola mobile

- E' utile perché
  - Permette di rappresentare in maniera compatta numeri molto grandi, ma anche molto piccoli, sia positivi sia negativi
- Numeri reali rappresentati tramite una coppia di numeri <m,e>

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

  - m: *mantissa* (detto anche *significante*), normalizzata tra due potenze successive della *base* b

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

  - e: *esponente* intero (detto anche *caratteristica*)
- Sia m che e hanno un numero fissato di cifre:

Intervalli limitati
Errori di arrotondamento

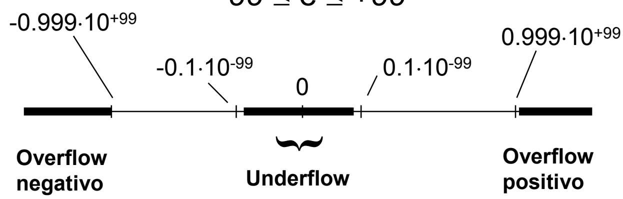
Esempio in base 10

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

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

- Esponente: 2 cifre con segno

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



- Notare che con lo stesso numero di cifre in notazione a virgola fissa ± XXX .YY
  - L'intervallo scende [-999.99,+999.99]
  - Ma si hanno 5 cifre significative invece di 3

Rappresentazione in virgola mobile dei numeri binari

- Lo stesso approccio della virgola mobile può essere seguito per rappresentare i numeri binari come $\pm m \cdot b^{\pm e}$

$$1.xxxxxxxxx_2 \cdot 2^{yyyy}$$

- Da notare che, di solito, la base b è implicita e quindi
  - È sufficiente memorizzare *segno*, *mantissa* (o significante) ed *esponente* (o caratteristica)
- Si usa un certo numero di bit (almeno 32)
  - Si riserva spazio per segno, mantissa ed esponente

Standard per la rappresentazione

- Importanza di definire uno standard per la rappresentazione dei numeri in virgola mobile
  - Per definire la semantica delle istruzioni in virgola mobile
- International Standard Organization (ISO)
  - Vi appartengono circa 100 organizzazioni che si occupano di creare standard
- IEEE Computer Society (Institute of Electrical and Electronics Engineers) definisce lo “IEEE standard for binary floating arithmetic” (riferito come **IEEE 754**) nel 1985
  - Specifica il formato, le operazioni, le conversioni tra i diversi formati floating point e quelle tra i diversi sistemi di numerazione, il trattamento delle eccezioni
- Nel 1989 IEEE 754 diventa uno standard internazionale (IEC 559)

Standard IEEE 754

- Formato non proprietario, ossia non dipendente dall’architettura del calcolatore
- Precisione *semplice* a 32 bit:

1	8	23
+/–	esp	mantissa

- Precisione *doppia* a 64 bit:

1	11	52
+/–	esp	mantissa

- Notazioni in modulo e segno
- Alcune configurazioni dell’esponente sono riservate
- Segno (1 bit):
  - 0 per positivo, 1 per negativo

IEEE 754 a 32 bit: esponente

- Esponente (8 bit)
  - Rappresentato in eccesso 127 (*polarizzazione* o *bias*)
  - L'intervallo di rappresentazione è [-127, 128]
  - Le due configurazioni estreme sono riservate, quindi

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

  - Se gli 8 bit dell'esponente contengono 10100011 = 163_{10}
    - L'esponente vale 163-127=36
  - Se gli 8 bit dell'esponente contengono 00100111 = 39_{10}
    - L'esponente vale 39-127=-88

- Perché la polarizzazione?
  - Il numero più grande che può essere rappresentato è 11…11
  - Il numero più piccolo che può essere rappresentato è 00…00
  - Quindi, quando si confrontano due interi polarizzati, per determinare il minore basta considerarli come interi senza segno

IEEE 754 a 32 bit: mantissa

- Mantissa (23 bit)
  - E' sempre *normalizzata*
    - Convenzione che la prima cifra significativa si trovi immediatamente a sinistra del punto decimale
      - Si ottiene aumentando o diminuendo il valore dell'esponente di tante unità quante sono le posizioni di cui è spostato il punto, ad es. $128 \cdot 10^{12} = 1.28 \cdot 10^{14}$

  - Se ne rappresenta soltanto la parte frazionaria

  - Due rappresentazioni, a seconda del valore dell'esponente:
    1) Numeri *normalizzati*: e≠00000000
    2) Numeri *denormalizzati:* e=00000000

Numeri normalizzati

- Un numerale si intende in questa rappresentazione quando **e≠00000000**
- In questa rappresentazione, la mantissa è normalizzata tra 1 e 2: **1 ≤ m < 2**
- Quindi, la mantissa è sempre nella forma:

1.XXXXXXXXX…X

- Si usano tutti i 23 bit per rappresentare la sola parte frazionaria (1 prima della virgola è implicito)
- 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 è comunque inferiore
- L'intervallo $(-2^{-126}, 2^{-126})$ è detto *intervallo di underflow*

Numeri denormalizzati

- Un numerale si intende in questa rappresentazione quando **e=00000000**
- L'esponente assume il valore *convenzionale* -126
- La mantissa è normalizzata tra 0 e 1: **0** < **m < 1**
- Quindi, la mantissa è sempre nella forma:

0.XXXXXXXXX…X

- Si usano tutti i 23 bit per rappresentare la sola parte frazionaria
- La più piccola mantissa vale 2^{-23}
- Gli intervalli rappresentati sono:

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

NB *Più piccola è la mantissa, minore è il numero di cifre significative*

Esempi: conversione da virgola mobile

- Quale numero in singola precisione rappresentano i seguenti 32 bit

  1 10000001 01000000000000000000000
  - Segno negativo (-)
  - Esponente e = $2^7 + 2^0 - 127 = 129 - 127 = 2$
  - Mantissa m = $1 + 2^{-2} = 1.25$
  - Quindi il numero rappresentato è $-1.25 \cdot 2^2 = -5$

  0 10000011 10011000000000000000000
  - Segno positivo (+)
  - Esponente e = $2^7 + 2^1 + 2^0 - 127 = 131 - 127 = 4$
  - Mantissa m = $1 + 2^{-1} + 2^{-4} + 2^{-5} = 1.59375$
  - Quindi il numero rappresentato è $1.59375 \cdot 2^4 = 25.5$

Esempi: conversione in virgola mobile

- Quale è la rappresentazione a singola precisione del numero

  8.5
  - Segno positivo (0)
  - 8.5 in binario è $1000.1 \cdot 2^0 = 1.0001 \cdot 2^3$
  - Esponente e: 3 + 127 = 130 = 10000010
  - Mantissa m: 00010000000000000000000
  - Quindi 0 10000010 00010000000000000000000

  -13.75
  - Segno negativo (1)
  - 13.75 in binario è $1101.11 \cdot 2^0 = 1.10111 \cdot 2^3$
  - Esponente e: 3 + 127 = 130 = 10000010
  - Mantissa m: 10111000000000000000000
  - Quindi 1 10000010 10111000000000000000000

Standard IEEE 754 a 32 bit: estremi degli intervalli

- Più grande normalizzato: ~$\pm 2^{128}$

  X 11111110 11111111111111111111111

  $\pm$ 2^{127} ~2
- Più piccolo normalizzato: $\pm 2^{-126}$

  X 00000001 00000000000000000000000

  $\pm$ 2^{-126} 1
- Più grande denormalizzato: ~ $\pm 2^{-126}$

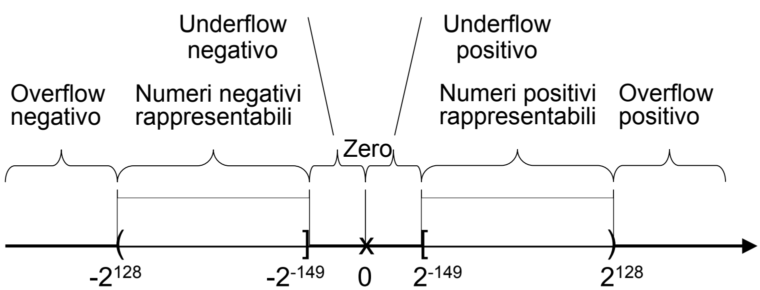
  X 00000000 11111111111111111111111

  $\pm$ 2^{-126} $(0.11...)_2 \approx 1$
- Più piccolo denormalizzato: $\pm 2^{-149}$

  X 00000000 00000000000000000000001

  $\pm$ 2^{-126} $(0.00...1)_2 = 2^{-23}$

Intervallo di rappresentazione



- L'overflow può essere positivo

  Quando si devono rappresentare numeri positivi maggiori di 2^{128}
- L'overflow può essere negativo

  Quando si devono rappresentare numeri negativi minori di $–2^{128}$
- L'underflow può essere positivo

  Quando si devono rappresentare numeri positivi minori di 2^{-149}
- L'underflow può essere negativo

  Quando si devono rappresentare numeri negativi maggiori di -2^{-149}

Confronto tra numeri in virgola mobile

- Per stabilire quale di due numeri in virgola mobile sia il maggiore
- Se sono di segno *discorde*, allora il numero positivo è maggiore
- Se sono di segno *concorde*
  - Se sono *positivi*
    - Il numero con l'esponente *più grande* è il maggiore; a parità di esponente, il numero con mantissa *più grande* è maggiore
  - Se sono *negativi*
    - Il numero con l'esponente *più piccolo* è il maggiore; a parità di esponente, il numero con mantissa *più piccola* è maggiore

Confronto tra numeri in virgola mobile (2)

- Per due numeri positivi (negativi)
  - Siano a e b due numeri positivi (negativi) rappresentati in virgola mobile da $a_{31}a_{30}\ldots a_0$ e $b_{31}b_{30}\ldots b_0$
  - Notare che $a_{31} = b_{31} = 0$ (1)
- Per verificare quale dei due sia il maggiore, non occorre nessuna conversione
  - È sufficiente confrontarli come se fossero interi senza segno
  - Basta scorrere i bit, ed al primo bit diverso si individua il maggiore
    - Il numero con l'i-esimo bit a 1 (0)
    - Il numero con esponente/mantissa più grande (più piccolo) è il maggiore

Configurazioni particolari

- Lo standard IEEE 754 attribuisce valori convenzionali a particolari configurazioni di *e* ed *m*
  - *e* ed *m* tutti 0: rappresentano il valore 0 (*altrimenti non rappresentabile*)
  - *m* tutti 0 ed *e* tutti 1: rappresentano l'*infinito* ($\pm\infty$)
  - $m \neq 0$ ed *e* tutti 1: rappresentano la situazione *Not A Number (NAN)*, cioè un valore indefinito (ad es. il risultato di una divisione per 0 o la radice quadrata di un numero negativo)

- Queste convezioni sono una caratteristica peculiare della notazione IEEE 754; non valgono, se non esplicitamente definite, per altre notazioni

Osservazioni sulla precisione singola

- In modo assolutamente indipendente dalla rappresentazione usata, con 32 bit è possibile rappresentare "soltanto" 2^{32} valori diversi
- I numeri rappresentati in virgola mobile hanno una densità maggiore vicino allo zero
- Diversi compromessi nella scelta del formato
- Incrementando la dimensione dell'esponente
  - Si diminuisce la dimensione del significando
  - Si espande l'intervallo di rappresentazione (esponenti maggiori) ma si perdono cifre significative (precisione)
- Possibilità di usare altre basi implicite (no standard IEEE 754!)
  - IBM S/390 usa base 16: aumenta l'intervallo di rappresentazione, a scapito della precisione

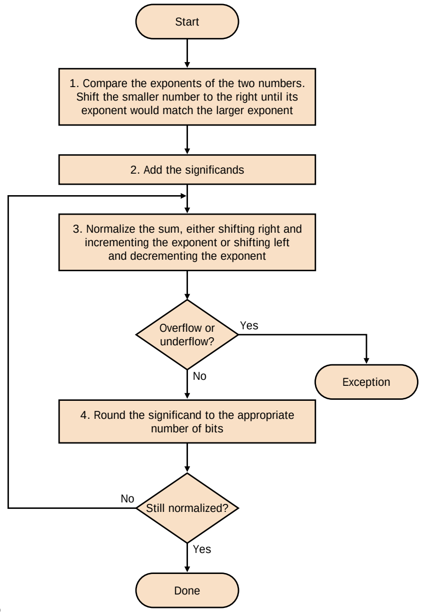
IEEE 754 a 64 bit

- Segno (1 bit)
- Esponente (11 bit)
  - Rappresentato in eccesso 1023
  - L'intervallo di rappresentazione è [-1023, 1024]
- Mantissa (52 bit)
  - Normalizzata come nella singola precisione
- Configurazione riservate come nella singola precisione per la rappresentazione di
  - 0
  - Numeri denormalizzati (positivi e negativi)
  - Infinito (positivo e negativo)
  - NaN (Not a Number)
- Esercizio
  - Quali è l'intervallo di rappresentazione dei numeri a doppia precisione?

Altri formati dello standard IEEE 754

- Oltre ai formati base per la singola precisione (32 bit) e la doppia precisione (64 bit), sono stati definiti i formati estesi
  - A singola precisione
  - A doppia precisione

Addizione in virgola mobile



Addizione in virgola mobile (2)

- Per addizionare e sottrarre occorre scalare le mantisse per eguagliare gli esponenti
- Esempio (*Notazione IEEE 754*)

  $n_1 + n_2$

  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

  Bit implicito della mantissa

  n'_1 : 0 10101010 00000000000000001000101 +

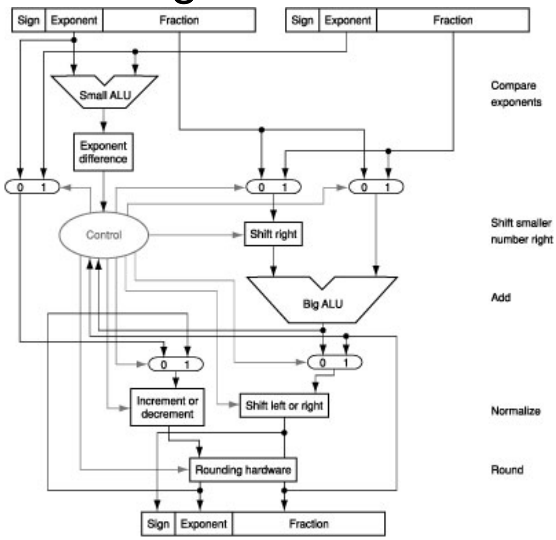
  n_2 : 0 10101010 11001100111000111000100

  0 10101010 11001100111001000001001 Somma delle mantisse

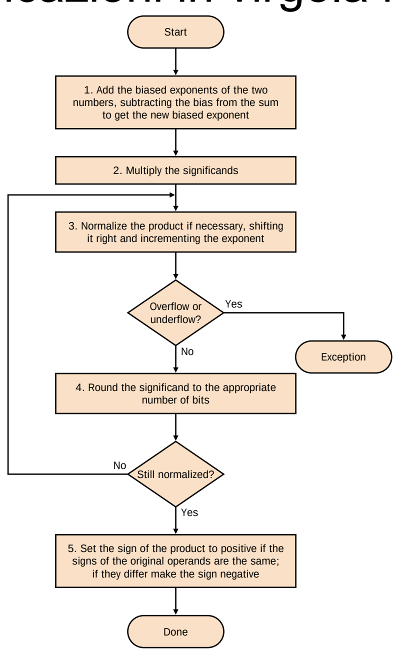
  0 10101010 11001100111001000001010 Round

- Notare che l'addendo più piccolo perde cifre significative

Unità aritmetica per addizione in virgola mobile



Moltiplicazioni in virgola mobile



Moltiplicazioni in virgola mobile (2)

- Si moltiplicano le mantisse e si sommano algebricamente gli esponenti
- Se necessario si scala la mantissa per normalizzarla e si riaggiusta l'esponente
- Esempio (*Notazione IEEE 754*)

$n_3 = n_1 \times n_2$

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
- $m_1 \times m_2$ = 10.0110001100101100001101**0**

N.B.: ricordare il bit implicito delle mantisse nella moltiplicazione

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

n_3 : 1 11000101 00110001100101100001101

Modalità di arrotondamento

- Lo standard IEEE 754 prevede 4 modalità di arrotondamento
- Arrotondamento al valore pari più vicino (più usato)
  - Al valore la cui cifra meno significativa è pari (si ha così sempre 0 nella cifra meno significativa)
- Arrotondamento a zero (troncamento)
- Arrotondamento a $+\infty$
- Arrotondamento a $-\infty$

Errore assoluto ed errore relativo

- Rappresentando un numero reale n nella 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*: $e_R = e_A/n = (n-n')/n$
- Se la mantissa è normalizzata, l'errore relativo *massimo* è costante su tutto l'intervallo rappresentato, ed è pari ad un'unità sull'ultima cifra rappresentata
  - Esempio: 10 cifre frazionarie $e_R = 2^{-10}$
- Nella notazione non normalizzata, l'errore relativo massimo non è costante

Esempio 1

- Rappresentazione binaria in virgola mobile a 16 bit:
  - 1 bit per il segno (0=positivo)
  - 8 bit per l'esponente, in eccesso 128
  - 7 bit per la parte frazionaria della mantissa normalizzata tra 1 e 2
- Calcolare gli estremi degli intervalli rappresentati, i numerali corrispondenti, e l'ordine di grandezza decimale
- Rappresentare in tale notazione il numero *n* rappresentato in CP2 dai tre byte FF5AB9
- Calcolare l'errore relativo ed assoluto che si commette rappresentando *n* nella notazione data

Esempio 1 (2)

1	8	7
+/–	esp	mantissa

- Estremi degli intervalli e numerali

  $-128 \leq e \leq 127$
  - Numero più grande: $\sim 2^{128}$

    X 11111111 1111111

    $\pm$ $2^{255-128}=2^{127}$ ~ 2
  - Numero più piccolo: 2^{-128}

    X 00000000 0000000

    $\pm$ 2^{0-128} 1
  - Intervalli: $(-2^{128}, -2^{-128}]$ $[2^{-128}, 2^{128})$
- Ordini di grandezza decimale

  $2^{128} = 2^{3\cdot 40 + 8} = 2^8 (2^{40})^3 \sim 2^8 (10^{12})^3 = 256\cdot 10^{36} = 2.56\cdot 10^{38}$

Esempio 1 (3)

- Rappresentare nella notazione data il numero *n* rappresentato in CP2 dai tre byte FF5AB9

  $(FF5AB9)_{CP2}$ = 1111 1111 0101 1010 1011 1001 =

  = -(0000 0000 1010 0101 0100 0111) =

  $= -(2^{15}+2^{13}+2^{10}+2^{8}+2^{6}+2^{2}+2^{1}+2^{0})=$

  $= -2^{15}(2^{0}+2^{-2}+2^{-5}+2^{-7}+2^{-9}+2^{-13}+2^{-14}+2^{-15})$
  - Esponente

    $15+128 = (143)_{10} = (10001111)_2$
  - Mantissa

    0100101
  - La rappresentazione chiesta è quindi: 1 10001111 0100101
- Errore relativo ed assoluto
  - Errore assoluto e_A

    $e_A = n - n' = -2^{15}(2^{0}+2^{-2}+2^{-5}+2^{-7}+2^{-9}+2^{-13}+2^{-14}+2^{-15}) +$

    $+ 2^{15}(2^{0}+2^{-2}+2^{-5}+2^{-7}) = -2^{15}(2^{-9}+2^{-13}+2^{-14}+2^{-15}) = -(2^{6}+2^{2}+2^{1}+2^{0}) = -71$
  - Errore relativo e_R

    $e_R = e_A/n \sim 2^{-9}$ (vedi il rapporto tra e_A ed n)

Esempio 2

- Rappresentazione binaria in virgola mobile a 16 bit:
  - 1 bit per il segno (0=positivo)
  - 8 bit per l'esponente, in eccesso 128
  - 7 bit per la parte frazionaria della mantissa normalizzata tra 1 e 2
- Dato il numero razionale *m* rappresentato in tale notazione dai due byte 43A5, calcolare l'intero *n* che approssima *m* per difetto, e rappresentarlo in complemento a 2 con 16 bit

Esempio 2 (2)

- *m* = 43A5 = (0100 0011 1010 0101)
  0 10000111 0100101
  - Segno positivo (+)
  - Esponente: $2^7+2^2+2^1+2^0-2^7 = 7$
  - Mantissa: $1+2^{-2}+2^{-5}+2^{-7}$
  - $n = 2^7 \cdot (2^0+2^{-2}+2^{-5}+2^{-7}) = 2^7+2^5+2^2+2^0 = (165)_{10} =$
    $= (10100101)_2$
  - In CP2 con 16 bit: 0000000010100101

Esempio 3

- Rappresentazione binaria in virgola mobile a 16 bit:
  - 1 bit per il segno (0=positivo)
  - *e* bit per l'esponente, in eccesso 2^{e-1}
  - 15-*e* bit per la parte frazionaria della mantissa normalizzata tra 1 e 2
- Calcolare il valore minimo e_{min} di bit per l'esponente che consenta di rappresentare il numero *n* rappresentato in CP2 dai tre byte FF5AB9
- e_{min} per rappresentare *n* in CP2 dato da FF5AB9

  FF5AB9 = 1111 1111 0101 1010 1011 1001 =

  = -(0000 0000 1010 0101 0100 0111) =

  $= -(2^{15}+2^{13}+2^{10}+2^{8}+2^{6}+2^{2}+2^{1}+2^{0}) =$

  $= -2^{15}(2^{0}+2^{-2}+2^{-5}+2^{-7}+2^{-9}+2^{-13}+2^{-14}+2^{-15})$

  Esponente = 15

  8<15<16 da cui e_{min} = 5

Esempio 4

- Rappresentazione binaria in virgola mobile a 16 bit:
  - 1 bit per il segno (0=positivo)
  - 7 bit per l'esponente, in eccesso 64
  - 8 bit per la parte frazionaria della mantissa normalizzata tra 1 e 2
- Dati *m* e *n* rappresentati in tale notazione dalle stringhe esadecimali FA53 e E8F2
- Calcolare la somma di *m* e *n* e fornire la stringa esadecimale che la rappresenta nella notazione suddetta

Esempio 4 (2)

- 1 bit per il segno, 7 bit per l'esponente in eccesso 64, 8 bit per la parte frazionaria della mantissa

  m = FA53 = 1111 1010 0101 0011

  m = 1 1111010 01010011

  n = E8F2 = 1110 1000 1111 0010

  n = 1 1101000 11110010

  e_m = 1111010 = $2^6+2^5+2^4+2^3+2^1-64=58$

  e_n = 1101000 = $2^6+2^5+2^3-64=40$

- Occorre scalare m_n di 18 posizioni

  $m_{n'}$ = 00000000

  m = 1 1111010 01010011

  n' = 1 1111010 00000000

  1 1111010 01010011