Potenziamento in Informatica

Riccardo Ortale

Codifica dell'Informazione

CODIFICA BINARIA

- Codifica binaria: usa un alfabeto di **2** simboli
- Utilizzata nei sistemi informatici
  - Si utilizza una grandezza fisica (luminosità, tensione elettrica, la corrente elettrica), per rappresentare informazione
  - Si divide in due intervalli l'insieme dei valori che la grandezza può assumere: ogni intervallo corrisponde ad un simbolo
- Solo 2 simboli al fine di ridurre la probabilità di errore
  - Tanti più simboli si devono distinguere e tanto meno la rivelazione sarà affidabile (gli intervalli della grandezza fisica saranno meno ampi)

CONVERSIONE DECIMALE-BINARIO

- Si calcolano i resti della divisione per 2

18 : 2 = 9	resto 0
9 : 2 = 4	resto 1
4 : 2 = 2	resto 0
2 : 2 = 1	resto 0
1 : 2 = 0	resto 1

10010

137 : 2 =	68	resto 1
68 : 2 =	34	resto 0
34 : 2 =	17	resto 0
17 : 2 =	8	resto 1
8 : 2 =	4	resto 0
4 : 2 =	2	resto 0
2 : 2 =	1	resto 0
1 : 2 =	0	resto 1

10001001

CODIFICA DEI NUMERI INTERI

- Modulo e segno
  - Il bit più a sinistra rappresenta il segno del numero (0 = ‘+’ , 1 = ‘-’)
  - Esempio: +7 = 0111, -7 = 1111
  - Valori da $-2^{k-1}+1$ a $2^{k-1}-1$
  - Con k=4 bit: da $-2^3+1=-7$ a $2^3-1=+7$
  - Attenzione ci sono due zeri!
    +0=0000 e -0=1000

Complemento a 2

- **X** corrisponde al binario naturale di $\mathbf{2^k + X}$

  $+6_{dieci} \Rightarrow 2^4+6 = 22 \Rightarrow [1]0110 \Rightarrow 0110_{C2}$

  $-6_{dieci} \Rightarrow 2^4-6 = 10 \Rightarrow [0]1010 \Rightarrow 1010_{C2}$
- si rappresentano i valori da $\mathbf{-2^{k-1}}$ **a** $\mathbf{2^{k-1}-1}$
  - con 4 bit i valori vanno da –8 a +7
  - con 8 bit i valori vanno da –128 a +127
- Attenzione: c'è una sola rappresentazione dello 0
  - con 4 bit è $+0_{dieci} = 0000_{C2}$ mentre $1000_{C2} = -8_{dieci}$

Complemento a 2

Metodi alternativi per calcolare la rappresentazione di –X a partire da quella di X

- Effettuare il complemento di ogni bit di X e aggiungere poi 1
  - rappresentazione di $+6_{dieci} = 0110_{C2}$ (NB ci vogliono 4 bit!!)
  - complemento di tutti i bit $\Rightarrow 1001_{C2}$ (corrisponderebbe a -7_{dieci})
  - aggiungere 1 $\Rightarrow 1010_{C2}$ (che corrisponde a -6_{dieci})

Esercizio 1

- Convertire il numero decimale 111_{10} (111 decimale) in binario.
  - $111_{10} : 2_{10} = 55_{10}$ resto 1
  - $55_{10} : 2_{10} = 27_{10}$ resto 1
  - $27_{10} : 2_{10} = 13_{10}$ resto 1
  - $13_{10} : 2_{10} = 6_{10}$ resto 1
  - $6_{10} : 2_{10} = 3_{10}$ resto 0
  - $3_{10} : 2_{10} = 1_{10}$ resto 1
  - $1_{10} : 2_{10} = 0_{10}$ resto 1

- Quindi 111_{10} è equivalente a 1101111_2

Esercizio 2

- Convertire il numero 111_2 (111 binario) in decimale.
  - $111_2 = 1_{10} \times 2^2{}_{10} + 1_{10} \times 2^1{}_{10} + 1_{10} \times 2^0{}_{10} =$

    $= 4_{10} + 2_{10} + 1_{10} =$

    $= 7_{10}$
- Quindi 111_2 è equivalente a 7_{10}

Esercizio 3

- Convertire il numero 321_{10} in binario.
  - $321_{10} : 2_{10} = 160_{10}$ resto 1
  - $160_{10} : 2_{10} = 80_{10}$ resto 0
  - $80_{10} : 2_{10} = 40_{10}$ resto 0
  - $40_{10} : 2_{10} = 20_{10}$ resto 0
  - $20_{10} : 2_{10} = 10_{10}$ resto 0
  - $10_{10} : 2_{10} = 5_{10}$ resto 0
  - $5_{10} : 2_{10} = 2_{10}$ resto 1
  - $2_{10} : 2_{10} = 1_{10}$ resto 0
  - $1_{10} : 2_{10} = 0_{10}$ resto 1

- Quindi il numero 321_{10} è equivalente a 101000001_2.

Esercizio 4

- Convertire il numero 1010101_2 in decimale.
  - $1010101_2 = 1_{10} \times 2^6{}_{10} + 1_{10} \times 2^4{}_{10} + 1_{10} \times 2^2{}_{10} + 1_{10} \times 2^0{}_{10} = 64_{10} + 16_{10} + 4_{10} + 1_{10} = 85_{10}$
- Quindi il numero 1010101_2 è equivalente a 85_{10}.

Esercizio 5

- Convertire il numero -13_{10} in modulo e segno a 6 bit.
  - $13_{10} : 2_{10} = 6_{10}$ resto 1
  - $6_{10} : 2_{10} = 3_{10}$ resto 0
  - $3_{10} : 2_{10} = 1_{10}$ resto 1
  - $1_{10} : 2_{10} = 0_{10}$ resto 1
- Quindi il numero -13_{10} è equivalente a 101101_{MS} a 6 bit.

Modulo e segno: Si aggiungono in testa per arrivare a 6 bit. Il simbolo 1 iniziale rappresenta il segno -.

Esercizio 6

- Convertire i numeri 0111_{MS} e 1111_{MS} in decimale.
  - Poiché il primo bit di 0111_{MS} è zero, allora 0111_{MS} è un numero positivo e $0111_{MS}=0111_2$, di conseguenza $0111_{MS} = 7_{10}$ (vedi esercizio nr. 2).
  - Il primo bit di 1111_{MS} è 1, quindi 1111_{MS} è un numero negativo il cui valore assoluto è 111_2. In questo caso $1111_{MS} = -7_{10}$.

Esercizio 7

- Sommare i numeri 1011_2 e 10_2.

  1 0 1 1 +

  0 0 1 0 =

  \----------------

  1 1 0 1

- Quindi $1011_2 + 10_2$ è uguale a 1101_2.

Esercizio 8

- Convertire il numero -13_{10} in complemento a due a 8 bit.
  - Converti 13_{10} in binario a 8 bit → 00001101_2
  - Inverti i bit → 11110010_2
  - Somma 1_2 → 11110010_2 +

00000001_2 =

11110011_{C2}

- Quindi -13_{10} è equivalente a 11110010_{C2} a 8 bit.

Esercizio 9

- Convertire i numeri 0111_{C2} e 1111_{C2} in decimale.
  - Poiché il primo bit di 0111_{C2} è 0, si tratta di un numero positivo, quindi 0111_{C2}=0111_2, e di conseguenza 0111_{C2} = 7_{10} (vedi esercizio nr. 2).
  - Il primo bit di 1111_{C2} è 1, quindi 1111_{C2} è un numero negativo il cui valore assoluto si ottiene invertendo tutti i bit e sommando 1, come di seguito riportato:
  - Numero 1111_{C2} → 1111_{c2}
  - Inverti i bit → 0000_2
  - Somma 1_2 → 0000_2 +

    0001_2 =

    0001_2
- Quindi 1111_{C2} = -1_{10}.

Rappresentazione dei numeri reali

- 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): + XXX .YY
  - Notazione in virgola mobile
    - Dedica alcune cifre a rappresentare un esponente della base, che indica l'ordine di grandezza del numero rappresentato

La rappresentazione in virgola mobile

- *Il termine **numero in virgola mobile** indica il metodo di rappresentazione dei numeri razionali (e di approssimazione dei numeri reali) e di elaborazione dei dati usati dai processori per compiere operazioni matematiche.*
- *Si contrappone all'aritmetica intera o in virgola fissa. In informatica viene usata solitamente in base 2;*
  - *in questo caso può essere considerata l'analogo binario della notazione scientifica in base 10.*
  - *L'uso di operazioni aritmetiche in virgola mobile è ad oggi il metodo più diffuso per la gestione di numeri reali.*
- *Un numero in virgola mobile, è costituito nella sua forma più semplice da due parti:*
  - un campo di **mantissa** *m*;
  - un campo di **esponente** *e*.
- *In alcuni casi, ad esempio nello standard IEEE 754, si ha un ulteriore campo:*
  - *il **segno** s, ma ciò verrà trattato specificamente nelle slide relative.*

Perché la rappresentazione in virgola mobile

- Limitazioni della rappresentazione in virgola fissa
  - Fissato il numero di cifre e la posizione della virgola:
    - Non rappresenta bene numeri frazionari molto grandi
    - Non rappresenta bene numeri (frazioni) molto piccoli
  - Come rappresentare in virgola fissa:
    - 50000000000000.00003
    - 0.000000000000000008
- La rappresentazione in virgola mobile estende l'intervallo di numeri rappresentati a parità di cifre
  - 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

Perché la rappresentazione in virgola mobile

- *Questo metodo di scrittura permette di rappresentare un larghissimo insieme numerico all'interno di un determinato numero di cifre, cosa che la virgola fissa non concede.*
- *Un numero è caratterizzato*
  - *dal valore b, che costituisce la **base** della notazione in cui è scritto il numero,*
  - *e dalla quantità p di cifre presenti nella mantissa, detta **precisione**.*
- *La mantissa di un numero scritto con questo metodo si presenta quindi nella forma ±d.ddd...ddd (una quantità p di cifre d comprese tra 0 e b-1).*
  - *Se la prima cifra della mantissa non è zero, il numero è definito normalizzato.*
  - *Se viene usato il campo s, la mantissa deve essere positiva, e questo bit ne determina il segno.*

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 arrontondamento

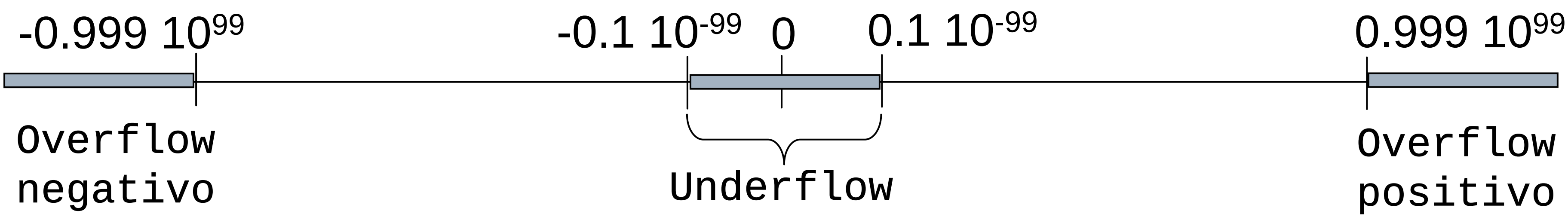
Esempio in base 10

- 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 $\pm m \cdot b^{\pm e}$
  - $\pm\, 1.xxxxxxxxx_2 \cdot 2^{\pm yyyy2}$
- 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 IEEE 754

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

1	8	23
±	esponente	mantissa

  - Precisione doppia a 64 bit:

1	11	52
±	esponente	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 = 3910
    - 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
  - Quando si confrontano due interi polarizzati,
    - per determinare il minore usando il bias li si considera come interi senza segno (ciò elimina la necessità di un bit per il segno)
    - in fase di decodifica dell'esponente, il bias deve essere sottratto per poter recuperare il valore originale

IEEE 754 a 32 bit

- I valori assunti dall'esponente e dalla mantissa m determinano l'appartenenza del numero ad una di queste categorie:
  - zeri;
  - numeri in forma normale;
  - numeri in forma denormalizzata;
  - infiniti;
  - Nan (*not a number*).

Numeri normalizzati

- *Il campo m è una stringa di bit che rappresenta la sequenza di cifre dopo la virgola.*
- *Tutte le mantisse sono normalizzate in modo che il numero prima della virgola sia 1.*
- *Pertanto, per un dato m il valore matematico corrispondente è* $M=1,m$
- *In pratica, la mantissa è costituita dal numero binario **1,** seguito dalla virgola e dalla parte intera del numero rappresentato, in forma binaria;*
  - *la mantissa risulta così artificialmente compresa tra 1 e 2.*
- *Quando un numero è normalizzato, come risulta dal suo esponente, il primo bit della mantissa, pari a 1, viene omesso per convenienza: viene quindi chiamato **bit nascosto**, o **bit implicito**.*

Numeri normalizzati

- Un numerale si intende in questa rappresentazione quando
  - **$e \neq 00000000$**
- In questa rappresentazione, la mantissa è normalizzata tra 1 e 2:

$$1 \leq m < 2$$

- 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
  - *Con questo sistema di rappresentazione, si hanno due zeri (+0 e −0) e due infiniti(+∞ e −∞) a seconda del valore del primo bit; e che i numeri subnormali possono avere un segno e una mantissa, utili però solo per l'analisi.*
- 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 tra 0 e 1: $\mathbf{0} < \mathbf{m} < \mathbf{1}$
- 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:

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

Esempi: conversione da virgola mobile

- Quale numero in singola precisione è rappresentato dai 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 $1.59375 \cdot 2^4 = 25.5$

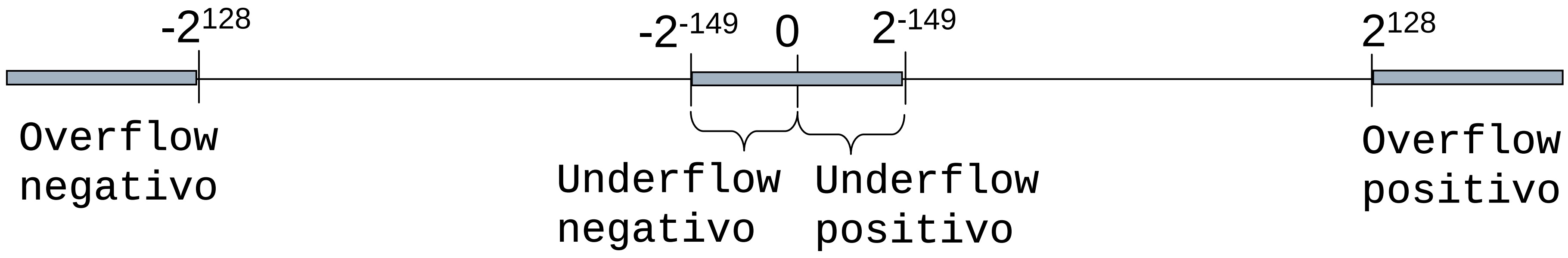
Esempi: conversione in virgola mobile

- 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: $\sim\pm 2^{128}$
  - X 11111110 11111111111111111111111
  - $\pm$ 2^{127} $(1.11\ldots)_2 \sim 2_{10}$
- Più piccolo normalizzato: $\pm 2^{-126}$
  - X 00000001 00000000000000000000000
  - $\pm$ 2^{-126} $(1.00\ldots)_2 = 1_{10}$
- Più grande denormalizzato: $\sim \pm 2^{-126}$
  - X 00000000 11111111111111111111111
  - $\pm$ 2^{-126} $(0.11\ldots)_2 \approx 1_{10}$
- Più piccolo denormalizzato: $\pm 2^{-149}$
  - X 00000000 00000000000000000000001
  - $\pm$ 2^{-126} $(0.00\ldots 1)_2 = 2^{-23}{}_{10}$

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
    - 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

- Per due numeri positivi (negativi)
  - Siano a e b due numeri positivi (negativi) rappresentati in virgola mobile da a31a30…a0 e b31b30…b0
  - Notare che a31 = b31 = 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 (±∞)
- *m* ≠ 0 ed *e* tutti 1: rappresentano la situazione *NotANumber* (NaN), cioè un valore indefinito (ad es. il risultato di una divisione per 0 o la radice quadrata di un numero negativo)

Normalizzati	±	exp≠0	Qualsiasi stringa di bit
Denormalizzati	±	0	Qualsiasi stringa diversa da 0
Zero	±	0	0
Infinito	±	111…1	0
NaN	±	111…1	Qualsiasi stringa diversa da 0

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)

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)

IEEE 754 a 64 bit

- Esercizio
  - Qual è l'intervallo di rappresentazione dei numeri a doppia precisione?

Esercizio 1

- *Convertire il numero 12.125 in binario.*
  - **Parte intera**
    - 12 (base 10) --> **1100 (base 2)**
  - **Parte decimale**
    - Si procede con moltiplicazioni successive.
      - 0.125 * 2 = 0.250 (riporto 0)
      - 0.250 * 2 = 0.500 (riporto 0)
      - 0.500 * 2 = 1.000 (riporto 1) – parte intera uguale a 1
      - 0.000 * 2 = FINE
    - Ordinando i riporti al contrario, si ottiene il risultato.
    - 0.125 (base 10) --> **0.001 (base 2)**
- Il risultato sarà: 12.125 (base 10) --> **1100. 001 (base 2)**

Esercizio 2

- *Convertire il numero 17.55 in binario (precisione 8 cifre decimali, arrotondamento per troncamento).*
- **Parte intera:** 17 (base 10) --> 10001 (base 2)
- **Parte decimale**
    - 0.55 * 2 = 1.10 (riporto 1)
    - 0.10 * 2 = 0.20 (riporto 0)
    - 0.20 * 2 = 0.40 (riporto 0)
    - 0.40 * 2 = 0.80 (riporto 0)
    - 0.80 * 2 = 1.60 (riporto 1)
    - 0.60 * 2 = 1.20 (riporto 1)
    - 0.20 * 2 = 0.40 (riporto 0)
    - 0.40 * 2 = 0.80 (riporto 0)
  - si sono ormai calcolate 8 cifre → si tronca il risultato
- Il risultato, è quindi pari a : **10001.10001100 (base 2) .**

Esercizi

- Convertire i seguenti numeri in binario, con una precisione di 8 cifre decimali.

Decimale	**Binario**
23.466	***10111.01110111***
61.625	***111101.10100000***
13.543	***1101.10001011***
55.110	***110111.00011100***
19.999	***10011.11111111***
22.001	***10110.00000000***
41.700	***101001.10110011***

Esercizio 3

- *Convertire il numero binario 1101.00101100 in decimale.*
  - Il procedimento è analogo alla conversione della sola parte intera.
  - **Parte intera:**
    - $1*2^0 + 0*2^{\wedge 1} + 1*2^{\wedge 2} + 1*2^{\wedge 3} = 1 + 0 + 4 + 8 = \mathbf{13}$
  - **Parte decimale:**
    - $0*2^{-1} + 0*2^{-2} + 1*2^{-3} + 0*2^{-4} + 1*2^{-5} + 1*2^{-6} + 0*2^{-7} + 0*2^{-8} =$ $0 + 0 + 0.125 + 0 + 0.03125 + 0.015625 + 0 + 0 =$ **0,171875**
- 1101.00101100 (Base 2) = **13.171875 (base 10)**

Esercizi

- *Convertire i seguenti numeri frazionari binari in decimale.*

Binario	**Decimale**
11101.11010111	*29.83984375*
10001.00001101	*17.05078125*
00011.10011001	*3.59765625*
11000.11111001	*24.97265625*
10011.00100100	*19.140625*
10010.01001001	*18.28515625*

Esercizio 4

- *Convertire il numero decimale 11.876 in rappresentazione reale Floating-Point (virgola mobile) su 32 bit. Indicare, in particolare, come vengono rappresentati mantissa ed esponente.*

Esercizio 4 (...continua...)

- **Mantissa**
  - La trasformazione prevede la conversione in binario del numero 11.876.
  - **Parte intera**
    - 11 (base 10) → **1011 (base 2).**
  - **Parte frazionaria**
    - Essendo la parte intera composta da 4 cifre, si dovranno calcolare 20 cifre decimali (la mantissa è composta da 24 cifre binarie).
    - 0.876 (base 10) → 0.11100000010000011000 (base 2, 20 cifre con approssimazione per troncamento)
  - La mantissa sarà pari a: 1011.11100000010000011000
  - Normalizzando, si avrà:1.01111100000010000011000 * 2^3
- Siccome il numero è positivo, e la prima cifra della mantissa si omette, la mantissa sarà pari a **01111100000010000011000.**

Esercizio 4 (...continua)

- **Esponente**
  - Avendo dovuto fare uno shift della mantissa di tre posizioni (si è moltiplicato per 2^3), l'esponente varrà 3.
  - Essendo però espresso in codice ECCESSO 127, il numero da inserire come esponente sarà
    - 127+3=130.
  - Si converte quindi questo numero in binario puro con i metodi già visti, ottenendo 10000010.

- Il risultato finale sarà:

Segno	**Esponente**	**Mantissa**
0	10000010	01111100000010000011000

Esercizi

- *Convertire i seguenti numeri decimali frazionari in codifica standard IEEE 754 SP.*

Decimale Segno Esponente Mantissa

-12.72 1 10000010
10010111000010100011101

+14.375 0 10000010
11001100000000000000000

-46.188 1 10000100
01110001100000010000011

+7.99 0 10000001
11111111010111000010100

-12.56 1 10000010
10010001111010111000010

+5.54 0 10000001
01100010100011110101110

-2.21 1 10000000
00011010111000010100011

FINE