Lezione 7
Standard IEEE-754
per le operazioni floating-point

http://www.dii.unisi.it/~giorgi/didattica/arcal1

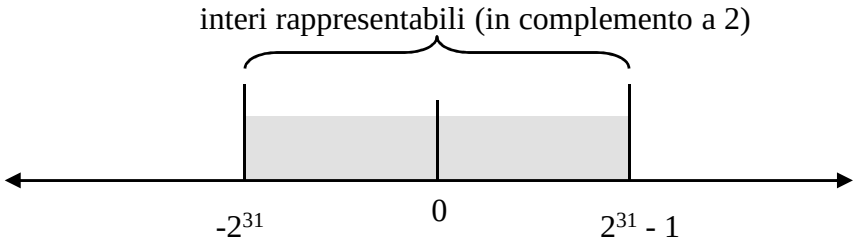
Floating-Point (breve schema)

- Nelle operazioni floating point dobbiamo rappresentare
  - numero con decimali, e.g., 3.1416
  - numeri molto piccoli, e.g., .000000001
  - numeri molto grandi, e.g., 3.15576×10^9
- Rappresentazione:
  - segno, mantissa, esponente: $(-1)^{segno} \times mantissa \times 2^{esponente}$
    - piu' bit di mantissa danno piu' accuratezza
    - piu' bit di esponente danno un intervallo piu' ampio
- IEEE 754 floating point standard:
  - singola precisione ("float" del C) → uso 32 bit per memorizzare:
    1 bit segno, 8 bit esponente, 23 bit mantissa**
  - doppia precisione ("double" del C) → uso 64 bit per memorizzare:
    1 bit segno, 11 bit esponente, 52 bit mantissa**

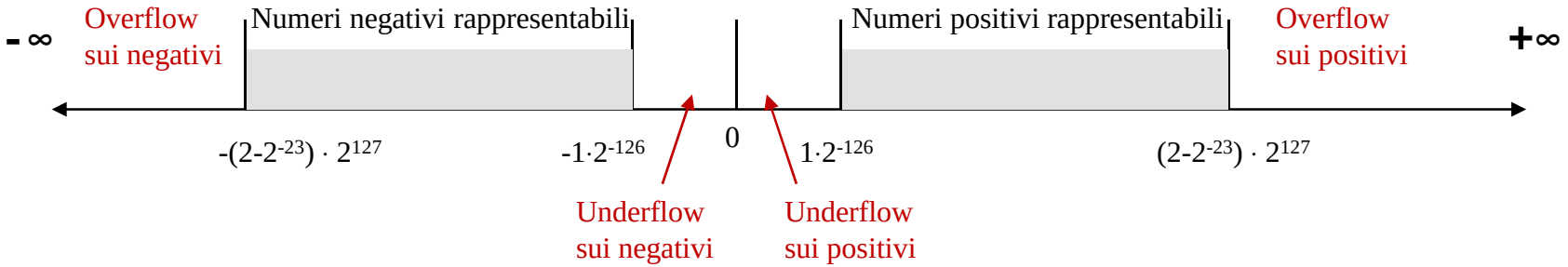
**Nota: utilizzando il "trucco" della normalizzazione della mantissa nella forma 1.xxxxx le cifre (binarie) effettivamente rappresentate sono rispettivamente 24 e 53

Intervalli rappresentabili

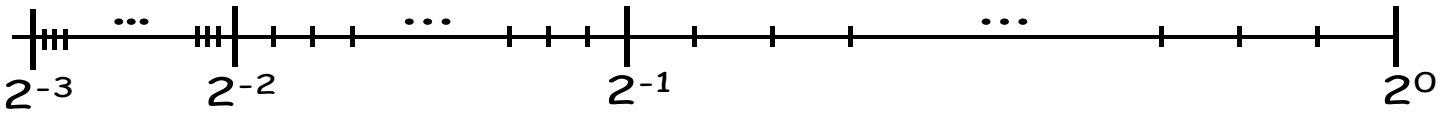
- **Nel caso di interi a 32 bit:**



- **Nel caso di floating-point a 32 bit (IEEE-754)**

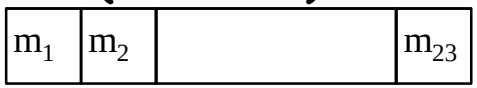


Nota: il numero di valori rappresentati con 32 bit e' circa lo stesso nel caso INT e in quello FP... la densita' dei punti non e' costante, quindi.



IEEE 754 floating-point standard

- **Mantissa**
  - Rappresentazione in base due con Normalizzazione (e modifica dell'esponente) in modo tale da riportare un 1 nella cifra piu' significativa
    - (tale unico) 1 prima della virgola non compare nella codifica
  - I bit dalla mantissa rappresentano un numero decimale (in base 2) >=1 e <2
    - M=1-mantissa= $m_1 \times 2^{-1} + m_2 \times 2^{-2} + \ldots + m_{23} \times 2^{-23}$



- **Esponente**
  - Rapprentazione "polarizzata" (biased) per rendere piu' facile il sort
    - polarizzazione=127 per la singola precisione, 1023 per la doppia precisione
    - i valori vanno da 1 a 2^e-2 (e=#bit per l'esponente) e codificano gli esponenti da $-2^{e-1}+2$ a $+2^{e-1}-1$ (e.g. -126..+127)
  - Codifiche riservate
    - esponente con tutti 1 e mantissa != 0
      codifica "Not a Number" (NAN) (risultato di 0/0)
    - esponente con tutti 1 e mantissa == 0
      codifica "Infinity" (INF) (risultato di un numero diverso da zero diviso 0)
    - esponente con tutti 0 e mantissa != 0
      codifica "underflow" (numero denormalizzato)

- **Ricapitolando: per $(-1)^S \times (1+M) \times 2^{E-polarizzazione}$ → memorizzo <S,M,E>**
  - S=0 o 1, M=mantissa_normalizzata senza "1." iniziale, E=esponente+polarizzazione

Esempio

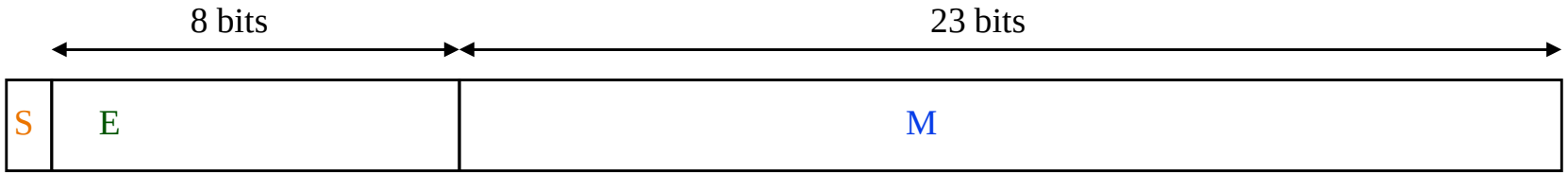
- Rappresentare -0.75 in formato IEEE single precision
  - rappresentazione decimale: $-0.75 = -3/4 = -3 \times 2^{-2}$
  - rappresentazione binaria: $-11 \times 2^{-2} = -1.1 \times 2^{-1}$
  - rappresentazione binaria normalizzata: -1.1×2^{-1}

  - segno=negativo
    → S=1
  - mantissa_normalizzata=1.1000 0000 0000 0000 0000 000
    → M = 1000 0000 0000 0000 0000 000
  - esponente=-1
    → E=esponente+polarizzazione = -1 + 127 =126 → 01111110

  - IEEE single precision: 10111111010000000000000000000000

  - Nota: la mantissa e' stata normalizzata. Per questo tutti i bit piu' significativi risultano spostati verso SINISTRA.

Floating-Point: single precision (32 bits)



Esempi con mantissa ed esponente positivi/negativi:

$+1.1010001 \cdot 2^{+10100}$ → 0 10010011 10100010000000000000000
$-1.1010001 \cdot 2^{+10100}$ → 1 10010011 10100010000000000000000
$+1.1010001 \cdot 2^{-10100}$ → 0 01101011 10100010000000000000000
$-1.1010001 \cdot 2^{-10100}$ → 1 01101011 10100010000000000000000

Esempi di numeri "notevoli":

0 → 0 00000000 00000000000000000000000
1 → 0 01111111 00000000000000000000000
minore rappresentabile (2^{-126}) → 0 00000001 00000000000000000000000
maggiore rappres. $(2-2^{-23}) \cdot 2^{127}$ → 0 11111110 11111111111111111111111
∞ → 0 11111111 00000000000000000000000
NaN → 0 11111111 XXXXXXXXXXXXXXXXXXXXXXX

Nota: $2^{-126} \approx 1.18 \cdot 10^{-38}$, $(2-2^{-23}) \approx 3.4 \cdot 10^{38}$

Tabella di riepilogo codifica FP nei processori INTEL

Da: Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture, Sep. 2008.
URL: http://download.intel.com/design/processor/manuals/253665.pdf

Class		Sign	Biased Exponent	Significand	
				Integer	Fraction
Positive	+ ∞	0	11..11	1	00..00
	+ Normals	0 • • 0	11..10 • • 00..01	1 • • 1	11..11 • • 00..00
	+ Denormals	0 • • 0	00..00 • • 00..00	0 • • 0	11..11 • • 00..01
	+ Zero	0	00..00	0	00..00
Negative	- Zero	1	00..00	0	00..00
	- Denormals	1 • • 1	00..00 • • 00..00	0 • • 0	00..01 • • 11..11
	- Normals	1 • • 1	00..01 • • 11..10	1 • • 1	00..00 • • 11..11
	- ∞	1	11..11	1	00..00
NaNs	SNaN	X	11..11	1	0X..XX[2]
	QNaN	X	11..11	1	1X..XX
	QNaN Floating-Point Indefinite	1	11..11	1	10..00
	Single-Precision		← 8 Bits →		← 23 Bits →
	Double-Precision		← 11 Bits →		← 52 Bits →
	Double Extended-Precision		← 15 Bits →		← 63 Bits →

Note:
(1) il bit della parte intera e' implicito e non viene quindi memorizzato
(2) La parte frazionaria di SNaN deve essere diversa da zero con il bit piu' significativo pari a 0

SNaN= Signaling NaN (solleva eccezione)

QNaN=Quiet NaN (non solleva eccezione)

QNaN-FP= "INDETERMINATO" → nessuno dei due operandi era un NaN: lo ha generato l'operazione

I numeri denormalizzati possono essere generati ma non vengono memorizzati

E' prevista una modalita' DAZ =Denormal-Are-Zero

E' supportata una modalita' a 80 bit (non IEEE-754)

Complessita' del Floating Point

- Le operazioni sono leggermente piu' complesse (v. testo)
- Oltre alla condizione di overflow si puo' avere "underflow"
  - (codifica: esponente con bit tutti a zero) → numeri denormalizzati
- L'accuratezza puo' essere un grosso problema
  - IEEE 754 prevede 2 ulteriori bit: 1 *guard* bit e 1 *round* bit
  - 4 modalita' per effettuare l'arrotondamento (*round to+∞, round to-∞, round to 0, round to nearest representable*)
  - altre complessita' (es. numeri denormalizzati)
- Implementare lo standard puo' essere "ingannevole"
  - Non usare lo standard puo' essere anche peggio
    - vedere il testo per la descrizione dell'80x86 e del Pentium bug!

Esempi di arrotondamento

numero	r.to 0	r.to nearest	r.to +inf	r.to -inf
1.00000003	1.0	1.0	1.00000012	1.0
1.00000007	1.0	1.00000012	1.00000012	1.0
-1.00000003	-1.0	-1.0	-1.0	-1.00000012
-1.00000007	-1.0	-1.00000012	-1.0	-1.00000012

Ricapitolazione

- L'aritmetica del calcolatore e' limitata da precisione finita
- I pattern di bit non hanno particolare significato ma esistono standard per:
  - complemento a due
  - IEEE 754 floating point
- Le istruzioni determinano il "significato" dei pattern di bit
- Le prestazioni e l'accuratezza sono importanti: ci sono diverse complessita' nelle macchine reali
- Lo standard e' stato rivisto e il documento pubblicato nel Giugno 2008 come IEEE 754-2008
  http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4610935&isnumber=4610934
  - Fra le novita' il supporto per floating point a 128-bit
  - Aritmetica floating point decimale (oltre che binaria)

EPSILON di macchina (MACHEPS = ε_{mach})

- E' la differenza fra 1 e "1+"
  (1+ ≡ il piu' piccolo numero rappresentabile maggiore di 1)
  - E' anche chiamato "precisione di macchina"
  - Da' una maggiorazione dell'errore di arrotondamento
- Esempio, IEEE-754 Single-Precision

esponente=0 | sottointeso 1.

$1 \equiv (0\ 01111111\ 000\ 0000\ 0000\ 0000\ 0000\ 0000)_{IEEE\text{-}754\text{-}SP}$
$= (1.000\ 0000\ 0000\ 0000\ 0000\ 0000)_2$
$= (1)_{10}$

esponente=0 | sottointeso 1.

$1+ \equiv (0\ 01111111\ 000\ 0000\ 0000\ 0000\ 0000\ 0001)_{IEEE\text{-}754\text{-}SP}$
$= (1.000\ 0000\ 0000\ 0000\ 0000\ 0001)_2$
$\approx (1.000000119209289\ldots\ldots)_{10}$

esponente= -23 | sottointeso 1.

➔ $\varepsilon_{mach} = (0\ 01101000\ 000\ 0000\ 0000\ 0000\ 0000\ 0000)_{IEEE\text{-}754\text{-}SP}$
$= (0.000\ 0000\ 0000\ 0000\ 0000\ 0001)_2$
$= (2^{-23})_{10} \approx (0.000000119209289\ldots\ldots)_{10} \approx (1.19209289\ldots \cdot 10^{-7})_{10}$

- In generale, per un tipo floating-point rappresentato in base b con p cifre di mantissa (normalizzato con una cifra !=0 prima della virgola), ovvero con p-1 cifre dopo la virgola

➔ $\varepsilon_{mach} = b^{-(p-1)} = b^{1-p}$ (es. nel caso IEEE-754 single-precision p=24)

Precisazioni su MACHEPS

- Alcuni usano come definizione: "il piu' piccolo numero positivo rappresentabile, tale che aggiunto ad 1 lo rende diverso da 1"
  - tale definizione e' diversa dalla precedente perche' il valore viene a dipendere dalla modalita' di arrotondamento usata
- Nel caso di IEEE-754 Single-Precision:
  - Per arrotondamenti al successivo (round to even) questa e' $2^{-24} + 2^{-47}$ (poco piu' di meta' rispetto alla precedente definizione)
  - Per arrotondamenti verso $+\infty$ questo e' il piu' piccolo numero rappresentabile, ovvero 2^{-149} (un numero denormalizzato)
  - Per arrotondamenti verso $-\infty$ questo coincide con la precedente definizione
- Il seguente programma C non determina MACHEPS bensi' un numero distante non piu' di un fattore 2 da MACHEPS, usando una ricerca lineare

Calcolo di MACHEPS

```
#include <stdio.h>

int main( int argc, char **argv )
{
   float machEps = 1.0f;

   printf( "current Epsilon, 1 + current Epsilon\n" );
   do {
      printf( "%G\t%.20f\n", machEps, (1.0f + machEps) );
      machEps /= 2.0f;
      /* Se sommando l'epsilon successivo ho 1 --> ho trovato epsilon */
   }
   while ((float)(1.0 + (machEps/2.0)) != 1.0);

   printf( "\nCalculated Machine epsilon: %G\n", machEps );
   return 0;
}
```

MACHEPS Output

```
current Epsilon, 1 + current Epsilon
1            2.0000000000000000000
0.5          1.5000000000000000000
0.25         1.2500000000000000000
0.125        1.1250000000000000000
0.0625       1.0625000000000000000
0.03125      1.0312500000000000000
0.015625     1.0156250000000000000
0.0078125    1.0078125000000000000
0.00390625   1.0039062500000000000
0.00195312   1.0019531250000000000
0.000976562  1.0009765625000000000
0.000488281  1.0004882812500000000
0.000244141  1.0002441406250000000
0.00012207   1.0001220703125000000
6.10352E-05  1.0000610351562500000
3.05176E-05  1.0000305175781250000
1.52588E-05  1.0000152587890625000
7.62939E-06  1.0000076293945312500
3.81470E-06  1.0000038146972656250
1.90735E-06  1.0000019073486328125
9.53674E-07  1.0000009536743164062
4.76837E-07  1.0000004768371582031
2.38419E-07  1.0000002384185791015

Calculated Machine epsilon: 1.19209E-07
```

Nota: in MATLAB
"eps" o "eps('double')" restituisce 2^{-52}
"eps('single')" restituisce 2^{-23}

MACHEPS di alcune macchine

Da: Kalbasi, K., "Can you trust your computer? [digital arithmetic]," *Potentials, IEEE* , vol.9, no.2, pp.15-18, Apr 1990.
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=52995&isnumber=1912

Computer	Rounding or Chopping	base	cifre mantissa	Emin	Emax	MACHEPS
CDC CYBER 170	R	2	48	-976	1'071	3.55×10^{-15}
CDC CYBER 205	C	2	47	-28'626	28'718	1.42×10^{-14}
Cray-1	C	2	48	-8'192	8'191	7.11×10^{-15}
DEC VAX (single)	R	2	24	-127	127	5.96×10^{-8}
DEC VAX (double)	R	2	56	-1'023	1'023	1.11×10^{-16}
HP-11C, 15C	R	10	10	-99	99	5.00×10^{-10}
IBM 3033 (single)	C	16	6	-64	63	9.54×10^{-7}
IBM 3033 (double)	C	16	14	-64	63	2.22×10^{-16}
IBM/PC (single)	R	2	24	-126	127	5.96×10^{-8}
IBM/PC (double)	R	2	53	-1'022	1'023	1.11×10^{-16}
PRIME 850 (single)	C	2	23	-128	127	2.38×10^{-7}
PRIME 850 (double)	C	2	47	-32'896	32'639	1.42×10^{-14}

L'epsilon di macchina dipende da diversi fattori, incluso la modalita' di arrotondamento

FORMATO IEEE-754

Esercizio

- Compito 2/11/07 domanda 1