UNIVERSITÀ
degli STUDI
di CATANIA

Sistemi di numerazione posizionali, conversioni, standard IEEE 754.

Corso di programmazione I (A-E / O-Z) AA 2022/23

Corso di Laurea Triennale in Informatica

Fabrizio Messina
fabrizio.messina@unict.it

Dipartimento di Matematica e Informatica

Indice

Sistemi di numerazione posizionale e conversioni

Sistemi di numerazione posizionale

Nei sistemi di numerazione posizionale, ai simboli vene assegnato un "peso" in base alla **posizione** che occupano nella notazione.

$$[a_N \dots a_1 a_0] = \sum_{k=0}^{N} a_k B^k$$

Sistemi di numerazione posizionale

$$[a_N \dots a_1 a_0] = \sum_{k=0}^{N} a_k B^k$$

B è detta **base**.

B simboli $\{a_1, a_2, \dots, a_B\}$ usati per la rappresentazione.

Sistemi di numerazione posizionale

$$[a_N \dots a_1 a_0] = \sum_{k=0}^{N} a_k B^k$$

a_N è la cifra più significativa (Most Significant Digit), in quanto associata al peso maggiore.

a_0 è la cifra meno significativa (Least Significant Digit), in quanto associata al peso minore.

Sistemi di numerazione posizionale

$$[a_N \dots a_1 a_0] = \sum_{k=0}^{N} a_k B^k$$

Sistema **decimale**:

- $B = 10$ La base. Rappresenta anche il numero di cifre del sistema di numerazione.
- $a_k \in \{0, 1, \dots, 9\}$. L'insieme di simboli usato nel sistema decimale.

Esempio: $[137]_{\mathbf{10}} = 1 \times 10^2 + 3 \times 10^1 + 7 \times 10^0$

Sistemi di numerazione posizionale

$$[a_N \dots a_1 a_0] = \sum_{k=0}^{N} a_k B^k$$

Sistema **binario**:

- $B = 2$, $a_i \in \{0, 1\}$. Due simbolo.

Esempio:
$[101010]_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

Il sistema binario si usa per rappresentare le **informazioni nei calcolatori**.

La cifra 0 o 1 è detta Bit (**B**inary Dig**it**).

Sistemi di numerazione posizionale

$$[a_N \dots a_1 a_0] = \sum_{k=0}^{N} a_k B^k$$

Sistema **esadecimale**:

- $B = 16$. I dieci simboli del sistema decimale sono estesi con le lettere A-F.
- $a_k \in \{0, 1, \dots, 9, A, B, C, D, E, F\}$

$$[B34f0]_{16} = 11 \times 16^4 + 3 \times 16^3 + 4 \times 16^2 + 15 \times 16^1 + 0 \times 16^0$$

Conversioni

Nei calcolatori, **dati codificati** in base due.

Algoritmi di conversione:

- Conversione da base 2 a base 10 di numeri interi e di frazioni.
- Conversione da base 10 a base 2 di numeri interi e di frazioni.
- Conversione da base 2 a base 16 (e viceversa) di interi.

Conversione da base 2 a base 10.

A - Conversione da base 2 a base 10 di numeri interi.

Vale la formula ($B = 2$)

$$[a_N \ldots, a_1 a_0] = \sum_{k=0}^{N} a_k B^k$$

Quindi:

$$[\mathbf{10101010}]_2 = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + \\ + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = [170]_{10}$$

Conversione da base 2 a base 10.

B - Conversione da base 2 a base 10 di frazioni

Un parte frazionaria è un numero razionale inferiore all'unità.

Se $x < 1$, allora si avrà $\{a_k\}_{k=1}^{M}$ tale che $(B = 2)$:

$$x = \sum_{k=1}^{M} a_k B^{-k}$$

NB: il peso della prima cifra dopo la virgola è 2^{-1}.

Conversione da base 2 a base 10.

ES: per il numero $[\mathbf{101.0101}]_2$

$$(x > 1) \quad [\mathbf{101}]_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = \mathbf{5_{10}}$$

$$(x < 1) \quad [\mathbf{.0101}]_\mathbf{2} = 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} =$$

$$= \mathbf{0.3125}_{10}$$

Dunque $[\mathbf{101.0101}]_\mathbf{2} = [5.3125]_{10}$

Homework H7.1

Effettuare la conversione dei seguenti numeri da base 2 a base 10.

A: 10110111

B: 11100100.101

C: 10100111.001

Conversione da base 10 a base 2

C - Conversione da base 10 a base 2 di numeri interi.

Algoritmo dei resti successivi.

Sia A un numero la cui rappresentazione in base 10 sia nota.

Sia $A = [\ldots 0\ 0\ \ldots\ 0\ a_N\ a_{N-1}\ \ldots\ a_0]_2 = \sum_{k=0}^{\infty} a_k 2^k$ (ogni a_k è termine incognito)

(*) $A = a_0 + 2\sum_{k=0}^{\infty} a_{k+1} 2^k \Rightarrow a_0 = 1$ **se e solo se** A è dispari, in quanto l'espressione (*) è formata da a_0 ed un termine pari.

Quindi a_0 è il resto della divisione intera $A/2$.

Conversione da base 10 a base 2

Era: $A = a_0 + 2\sum_{k=0}^{\infty} a_{k+1}2^k$.

Osservazione: $\sum_{k=0}^{\infty} a_{k+1}2^k$ è il quoziente della divisione $A/2$.

Si procede quindi allo stesso modo per tale quoziente (sia Q_0):

$$Q_0 = \sum_{k=0}^{\infty} a_{k+1}2^k = a_1 + 2\sum_{k=0}^{\infty} a_{k+2}2^k$$

Se il termine a sinistra è dispari, allora $a_1 = 1$, altrimenti $a_1 = 0$.

Quindi a_1 è il resto della divisione $Q_0/2$

Si procederà con tali divisioni – che permettono di determinare i termini a_k – **finchè non si ottiene quoziente nullo**.

Conversione da base 10 a base 2

ESEMPIO

$136_{10} = \mathbf{10001000_2}$.

$$
\left|\begin{array}{rcrll}
136 : 2 & = & 68 & \text{r}=\mathbf{0} & \text{(LSD)} \\
68 : 2 & = & 34 & \text{r}=\mathbf{0} & \\
34 : 2 & = & 17 & \text{r}=\mathbf{0} & \\
17 : 2 & = & 8 & \text{r}=\mathbf{1} & \\
8 : 2 & = & 4 & \text{r}=\mathbf{0} & \\
4 : 2 & = & 2 & \text{r}=\mathbf{0} & \\
2 : 2 & = & 1 & \text{r}=\mathbf{0} & \\
1 : 2 & = & 0 & \text{r}=\mathbf{1} & \text{(MSD)}
\end{array}\right|
$$

Homework H7.2

Effettuare la conversione dei seguenti numeri da base 10 a base 2.

A: 124

B: 2047

C: 912

Conversione da base 10 a base 2

D - Conversione da base 10 a base 2 di frazioni

Se $x < 1$ è una frazione di cui si conosce la rappresentazione in base 10, allora la sua rappresentazione in base 2 sarà una sequenza $[a_1 a_2 \dots a_N \dots]$ tale che:

$$x = \sum_{k=1}^{\infty} a_k 2^{-k}$$

Conversione da base 10 a base 2

Moltiplicando x per 2 si avrebbe: $2x = a_1 + \sum_{k=1}^{\infty} a_{k+1} 2^{-k}$.

Inoltre è noto che $\sum_{k=1}^{\infty} 2^{-k} = 1$, quindi $\sum_{k=1}^{\infty} a_{k+1} 2^{-k} \leq 1$.

Di conseguenza $2x = a_1 + \sum_{k=1}^{\infty} a_{k+1} 2^{-k} \geq 1 \iff a_1 = 1$, ovvero

a_1 è il **riporto** della moltiplicazione di x per 2.

Conversione da base 10 a base 2

$$2x = a_1 + \sum_{k=1}^{\infty} a_{k+1} 2^{-k}$$

Il numero $x' = \sum_{k=1}^{\infty} a_{k+1} 2^{-k}$ rappresenta quindi la parte frazionaria del numero $2x$.

Moltiplicando tale frazione per due si ottiene:

$$2x' = a_2 + \sum_{k=1}^{\infty} a_{k+2} 2^{-k}$$

Dunque $2x' \geq 1 \iff a_2 = 1$.

Conversione da base 10 a base 2

Le moltiplicazioni si ripetono finchè non si verifica una delle seguenti condizioni:

- il risultato della **moltiplicazione** è **uno** (quindi la parte frazionaria è zero);
- si ottiene una frazione ottenuta in uno dei passi precedenti; in questo caso la **frazione di partenza non è rappresentabile con un numero finito di cifre** in base due.
  La frazione è un numero periodico in base due.

Conversione da base 10 a base 2

Esempio A (conversione da base 10 a base 2).

$[0.125]_{10} = [0.001]_2$

Infatti:

$$\left|\begin{array}{lcll} 0.125 \times 2 & = & 0.250 & \text{riporto } \mathbf{0} \\ 0.250 \times 2 & = & 0.500 & \text{riporto } \mathbf{0} \\ 0.500 \times 2 & = & 1.000 & \text{riporto } \mathbf{1} \\ 0.000 \times 2 & = & 0.000 & \text{FINE} \end{array}\right| \left|\begin{array}{l} a_1 = 0 \\ a_2 = 0 \\ a_3 = 1 \\ a_4 = a_5 = \ldots = 0 \end{array}\right.$$

Conversione da base 10 a base 2

Esempio B (conversione da base 10 a base 2).

$[0.55]_{10} = [0.10\overline{0011}]_2$ (numero periodico in base 2)

$$
\left|\begin{array}{lcll}
0.55 \times 2 & = & 1.10 & \text{riporto } \mathbf{1} \\
0.10 \times 2 & = & 0.20 & \text{riporto } \mathbf{0} \\
0.20 \times 2 & = & \mathbf{0.40} & \text{riporto } \mathbf{0} \\
0.40 \times 2 & = & 0.80 & \text{ripotto } \mathbf{0} \\
0.80 \times 2 & = & 1.60 & \text{riporto } \mathbf{1} \\
0.60 \times 2 & = & 1.20 & \text{riporto } \mathbf{1} \\
0.20 \times 2 & = & \mathbf{0.40} & \text{riporto } \mathbf{0} \\
\ldots & = & \ldots & \ldots
\end{array}\right|
$$

Conversione da base 10 a base 2

Se il **numero non è rappresentabile con un numero finito di cifre in base 2**, si è costretti **a troncare la rappresentazione ad un certo numero di cifre**. ES: la rappresentazione in base due del numero 0.55 nel caso precedente, è stata troncata alla sesta cifra:

$$\mathbf{0.55_{10} = 0.100011_2}$$

In generale è **necessario approssimare mediante troncamento quando il numero di cifre a disposizione non è sufficiente a rappresentare** in modo completo **la frazione**.

Tale aspetto sarà affrontato in dettaglio quando si parlerà delle specifiche *IEEE 754*.

Homework H7.3

Effettuare la conversione dei seguenti numeri da base 10 a base 2.

A: 124.78

B: 0.1

C: 567.24

Base 16.

Indirizzi di memoria (e non solo) generalmente rappresentati (per comodità) in **esadecimale**.

```
1   int main(){
2     int a;
3     printf("%p", &a);
4   }
```

```
1   $ ./a.out
2   $ 0x5593ee84fe70
```

Per convenzione, si usa il prefisso **0x** prima delle cifre del numero esadecimale.

Base 16.

Rappresentazione esadecimale è più compatta di equivalente in base 2.

0x5593*ee*84*fe*70 $= [0101010110001 \ldots 0000]_2$ (48 cifre!)

Si usano 16 simboli: **0 1 2 3 4 5 6 7 8 9 A B C D E F**

B=16	B=2	B=16	B=2
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Base 16.

Conversione da base 2 e base 16: si convertono in esadecimale i gruppi di 4 cifre che compone il numero in base 2.

ES:

$[100100]_2 = [00100100]_2$ (aggiunte 2 cifre 0 a sinistra)

si ha $[0010]_2 = [2]_{16}$ e $[0100]_2 = [4]_{16}$

Dunque

$[100100]_2 = [00100100] = [24]_{16} = 2 \times 16^1 + 4 \times 16^0 = [36]_{10}$

Base 16.

Conversione da base 16 a base 2: si converte in base 2 ogni cifra esadecimale.

ES: il numero in base 16 $\mathbf{0x}f3c0$. Si ha

$[f]_{16} = [1111]_2$

$[3]_{16} = [0011]_2$

$[c]_{16} = [1100]_2$

$[0]_{16} = [0000]_2$

Dunque $\mathbf{0x}f3c0 = [1111001110100000]_2$.

Rappresentazione di numeri interi nei calcolatori.

Rappresentazione dei numeri INTERI: premessa

Tipicamente, per la rappresentazione dei numeri nei calcolatori si impiegano **sequenze di bit di lunghezza variabile** (8 (byte), 16 (word), 32 (double word) , 64, ...).

Numeri interi

Per rappresentare gli **interi (con o senza segno)**, i bit si impiegano per rappresentare:

- il **valore assoluto** (o modulo) del numero stesso.
- Eventuale **segno**. Varie soluzioni per la rappresentazione dei numeri negativi.

Overflow

- OVERFLOW: il modulo o valore assoluto di un certo numero **non è rappresentabile in quanto maggiore della massima grandezza rappresentabile** con il numero di bit a disposizione.

Quando si incorre in overflow ?

- Operazioni aritmetiche (somma, prodotto, etc)
- Operazione di assegnamento operate dal programmatore..

Overflow

ES: Somma..

	(riporto)	
	0 1 1 0 1 0 1 0	
A	0 1 1 0 1 0 1 0	+
B	0 0 1 0 1 0 1 1	
	1 0 0 1 0 1 0 1	

	(riporto)	
	1 1 1 0 1 0 1 0	
A	1 1 1 0 1 0 1 0	+
B	0 0 1 0 1 0 1 1	
	0 0 0 1 0 1 0 1	

(overflow!)

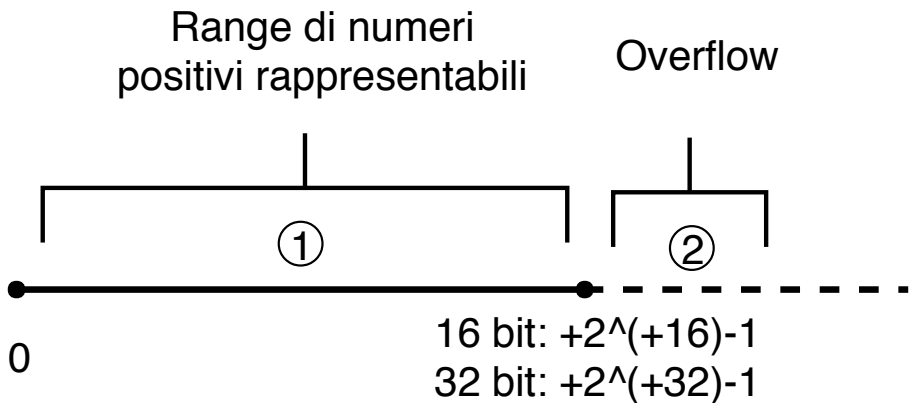
Riporto più a sinistra vale 1!

Rappresentazione di numeri interi senza segno

Codifica di interi **senza segno** con 16 bit:
Intervallo numerico: $[0, 2^{16} - 1] = [0, 65.535]$

Codifica di numeri interi **senza segno** con 32 bit:
Intervallo numerico: $[0, 2^{32} - 1] = [0, 4294967295]$

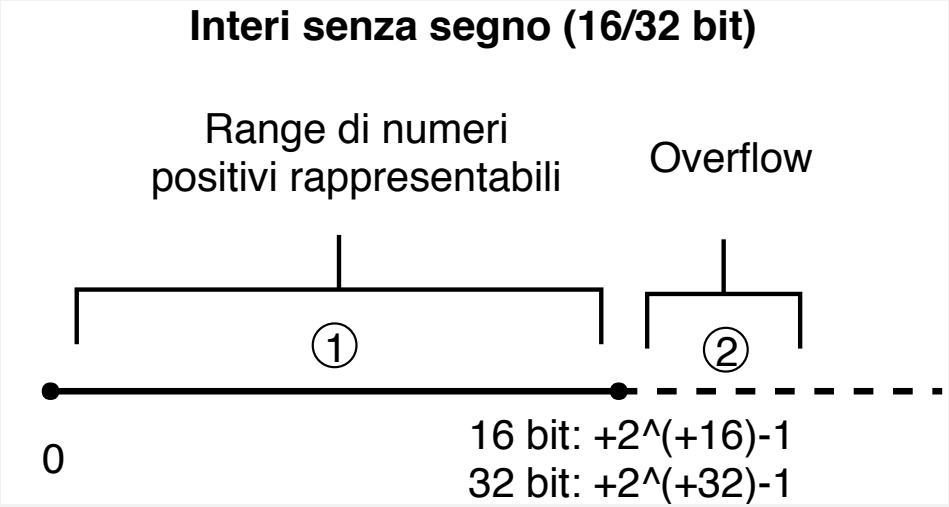
Interi senza segno (16/32 bit)



Overflow

Overflow: Il risultato numerico di una certa espressione aritmetica è **maggiore**, in valore assoluto, del valore massimo rappresentabile.

Overflow per interi senza segno

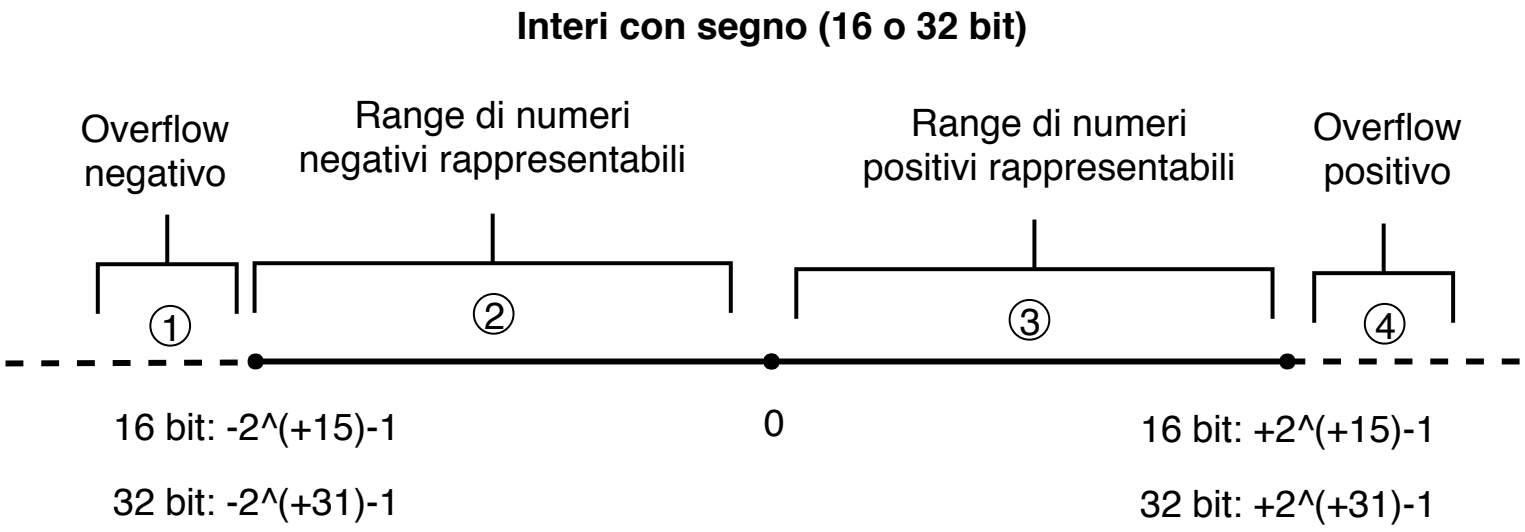


Rappresentazione di numeri Interi con segno

Rappresentazione con 1 bit per il segno (non impiegata nei calcolatori)

Codifica di numeri interi **con segno** a 16 bit:
Intervallo numerico $\pm(2^{15}-1) = 32767$ (1 bit per il segno)

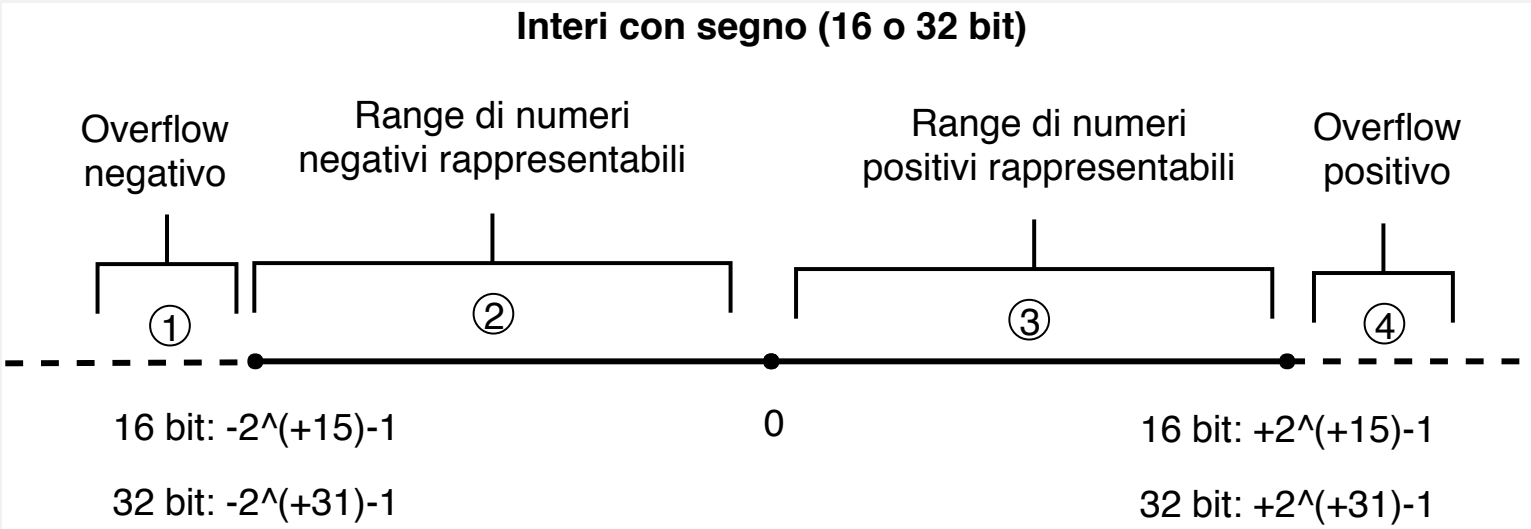
Codifica di numeri interi **con segno** a 32 bit:
Intervallo numerico $\pm(2^{31}-1) = 2.147.483.647$ (1 bit per il segno)



Rappresentazione di numeri Interi con segno

Overflow “negativo” e “positivo”..

Overflow per interi con segno



(Rappresentazione con 1 bit per il segno)

Esempio E7.1

Si immagini di avere a disposizione **8 bit** (MSB = Most Significant Bit).

$[123]_{10} = [01111011]_2$ (MSB $= 0$).

$[-123]_{10} = [11111011]_2$ (MSB $= 1$).

$[-130]_{10}$ non è rappresentabile con tale codifica e 8 bit, in quanto $130 > 2^7 - 1$

NB: le seguenti due codifiche...?

$[10000000]_2$ (MSB $= 1$), $[00000000]_2$ (MSB $= 0$).

Rappresentazione in COMPLEMENTO A DUE

Nella rappresentazione in **complemento a due**, il bit più significativo (MSB) assume **peso negativo**.

Dunque la sequenza $[a_{N-1}a_{N-2}\ldots a_0]_{2c}$ codifica il seg. valore:

$$A = -a_{N-1}2^{N-1} + \sum_{k=0}^{N-2} a_k 2^k$$

Valore minimo rappresentabile: $\mathbf{-2^{N-1}}$, quando $a_{N-1} = 1$ e $a_k = 0$ per ogni $k <= N-2$.

Valore massimo rappresentabile: $\mathbf{+2^{N-1}-1}$. Si ha quando $a_{N-1} = 0$ e $a_k = 1$ per ogni $k <= N-2$.

(Interi con segno: complemento a due)

Esempio.

Rappresentare il numero $[-99]_{10}$ in complemento a due con una stringa di 8 bit.

- Dato che il numero è negativo, il MSB deve essere uguale a 1.
- Quindi basta rappresentare, con i rimanenti 7 bit, il numero x tale che $2^{8-1} - x = 2^7 - x = 128 - x = 99$.
- Quindi $x = 29$, inoltre $[29]_2 = [0011101]_2$
- Dunque $-99_{2c} = 10011101$.

Rappresentazione di numeri Interi con segno

(Interi con segno: complemento a due). HOMEWORK H7.4

Rappresentare i seguenti numeri interi (con segno) mediante sequenze di 8 bit e codifica in complemento a due.

1. -170
2. $+98$
3. -67

Rappresentazione di numeri Interi con segno

(Interi con segno: complemento a due)

Osservazioni.

1. Dato che $2^{N-1} > 2^{N-1} - 1$, allora **quando il MSB è pari ad uno**, il **numero è negativo** qualunque siano i valori delle altre cifre;
   - $\rightarrow$ Esiste una **unica rappresentazione dello zero** (lo "zero positivo"!), NON +0 e -0, come precedente esempio 7.1...
2. Le operazioni di **somma e sottrazione** comodamente "unificate" (si vedrà nella prossima slide)..

Rappresentazione di numeri Interi con segno

(Interi con segno: complemento a due)

Una proprietà interessate di questa rappresentazione è costituita dalla **possibilità di invertire il segno di un numero** con due semplici operazioni: i) **inversione dei bit** e ii) **somma**.

Sia
$A = [110101]_{2c} = -1 \times 2^5 + 1 \times 2^4 + 1 \times 2^2 + 1 \times 2^0 = [-11]_{10}$.

i) Invertendo i bit si ha:

$A' = [001010]$.

ii) Infine si esegue la somma $A' + 1$
$A' + 1 = [001010] + [000001] = [001011]_{2c} = \cdots = 11_{10}$.

In tal modo operazioni di somma e sottrazione risultano "unificate" come operazioni di somma

Rappresentazione di numeri Interi con segno

(Interi con segno: complemento a due)

Casi di **overflow**.

- I due **operandi** hanno **segno differente**. NO Overflow! Non può verificarsi (Vedi precedente osservazione no.1)..
- Se i **due operandi** hanno **eguale segno**, allora potrebbe verificarsi overflow, positivo o negativo..

Come **riconoscere overflow** nell'operazione di somma/sottrazione?

- Ricordiamo che nella rappresentazione in complemento a due, le sottrazioni sono riconducibili alle somme...
- Esiste (errore) di overflow se e solo se i riporti nelle colonne di posto (n) ed $(n+1)$ **sono differenti**

Interi con segno: complemento a due (overflow)

$\boxed{-17 + 81}$

	(riporto)								
	1	**1**	1	1	1	1	1	1	
A		1	1	1	0	1	1	1	+
B		0	1	0	1	0	0	1	
	0	1	0	0	0	0	0	0	

Riporti colonne n ed $n+1$ identici!

$[01000000]_2 = [64]_{10} = 81 - 17$

$\boxed{-96 + (-64)}$

	(riporto)								
	1	**0**	0	0	0	0	0	0	
A		1	0	1	0	0	0	0	+
B		1	1	0	0	0	0	0	
	0	1	1	0	0	0	0	0	

Riporti colonne n ed $n+1$ differenti!

$[01100000]_2 = [96]_{10} \neq -160$.

Infatti -160 non è rappresentabile con 8 bit in complemento a due!

Interi con segno: complemento a due. HOMEWORK H7.5

Eseguire le seguenti operazioni in complemento a due:

$[00100010]_{2c} + [01110111]_{2c}$.

$[11111001]_{2c} - [00110101]_{2c}$.

$[00111001]_{2c} - [00011101]_{2c}$.

Numeri in virgola mobile. Standard IEEE 754

Rappresentazione dei numeri REALI

Nei calcolatori, i **numeri reali** sono rappresentati in un formato detto in **virgola mobile** (floating point).

Si tratta di una rappresentazione compatta che deriva dalla rappresentazione scientifica.

UNDERFLOW ($x \in \mathbb{R}, x < 1$): il modulo o valore assoluto di un certo numero **non è rappresentabile in quanto minore della minima grandezza** rappresentabile con il numero di bit a disposizione.

Esempio in base 10

a) $96.103 = 0.96103 \times 10^{+2}$

b) $2.96 = 0.296 \times 10^{+1}$

c) $2.96 = 29.6 \times 10^{-1}$

1) 0.96103, 0.296 e 29.6 sono denominati **mantissa** o **significando**.

2) 10 è la **base**.

3) **+2** e **+1** e **-1** sono denominati **esponente**.

Esempio in base 10

a) $96.103 = 0.96103 \times 10^{+2}$

b) $2.96 = 0.296 \times 10^{+1}$

c) $2.96 = 29.6 \times 10^{-1}$

Osservazione: i numeri b) e c) sono uguali, cambia solo la posizione della virgola, che si dice "flottante", ecco perchè il termine floating point.

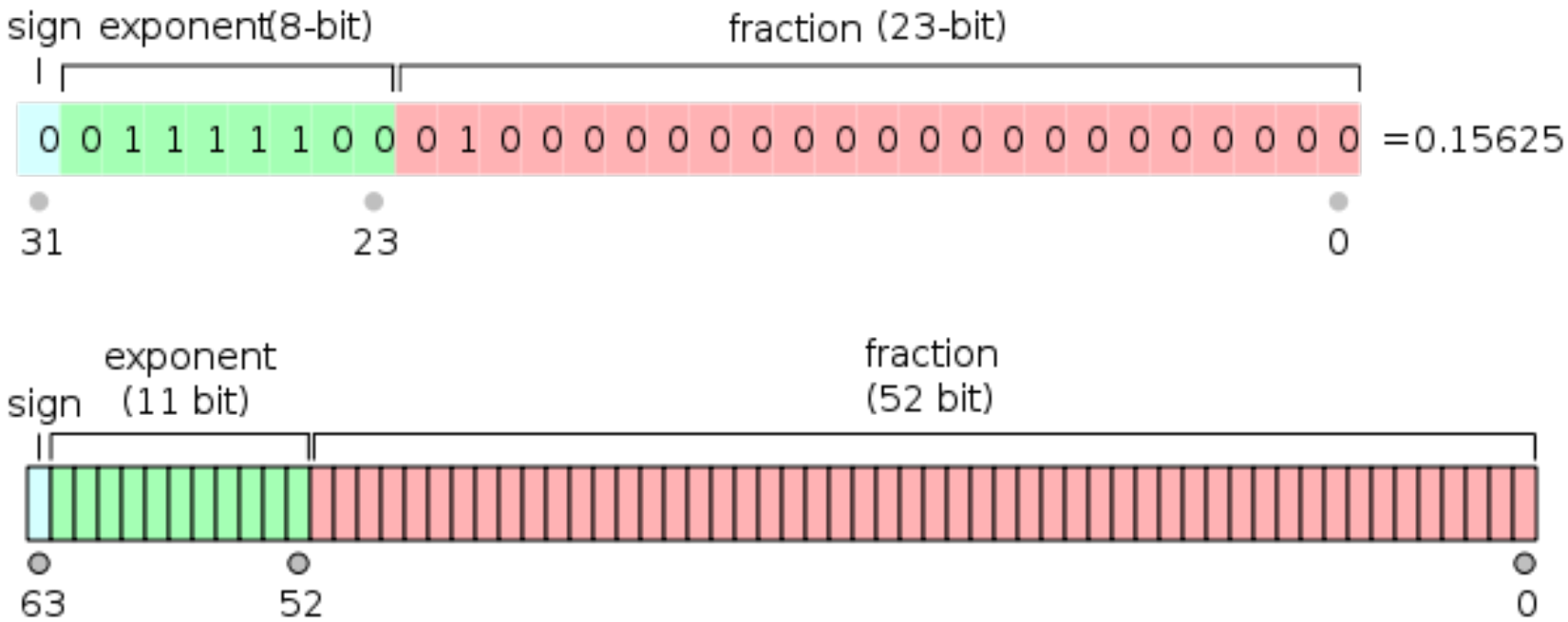
Standard IEEE 754.

Lo standard IEEE 754 definisce il formato per la rappresentazione dei numeri in virgola mobile, avendo a disposizione word di 32 o 64 bit:

- 1 bit per la rappresentazione del **segno** (s);
- 8 o 11 bit per la rappresentazione dello **esponente** (E) (kè una costante detta bias);
- 23 o 52 bit per la rappresentazione del **significando** o **mantissa** (M);

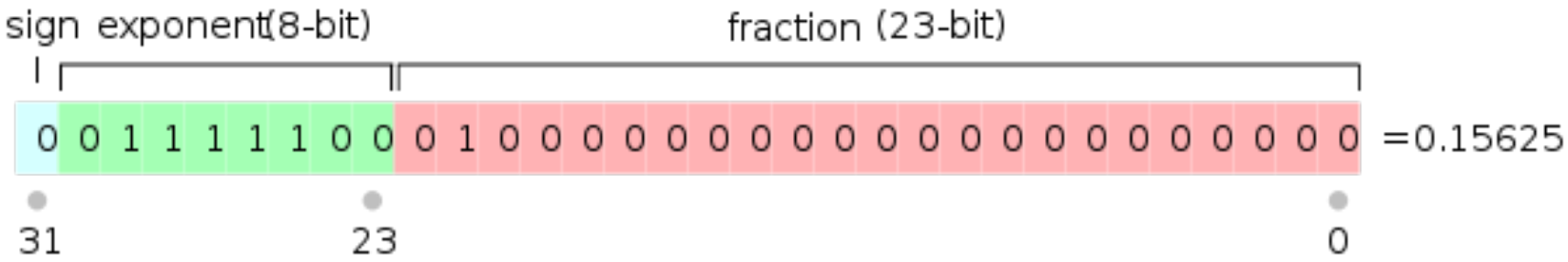
$$N = (-\mathbf{1})^{\mathbf{s}} \times \mathbf{2}^{\mathbf{E}-\mathbf{k}} \times \mathbf{1}.\mathbf{M}$$

IEEE 754



NB: “fraction” è la mantissa o significando.

Standard IEEE 754: Calcolo e rappresentazione della mantissa.



Mantissa o significando rappresentati in forma **normalizzata**:

- **si moltiplica o si divide** la codifica binaria della mantissa **per una certa potenza di 2**.
- In tal modo rappresentazione della mantissa rimarrà solo una cifra prima della virgola, cioè 1.
- Inoltre, dato che **la cifra prima della virgola** è sempre 1, questa **non viene rappresentata**, risparmiando cosi 1 bit.

Standard IEEE 754: Calcolo e rappresentazione della mantissa.

Esempio

Calcolo della mantissa o significando in formato **IEEE 754** a singola precisione (23 bit) per il numero $-\mathbf{113.25_{10}}$

1. Si calcola la **codifica** in base 2 del **valore assoluto** del numero: $113.25_{10} = 1110001.01_2$.
2. Per **normalizzare**, spostiamo la virgola di 6 posti : $1110001.01 = 1.11000101 \times 2^{+6}$.

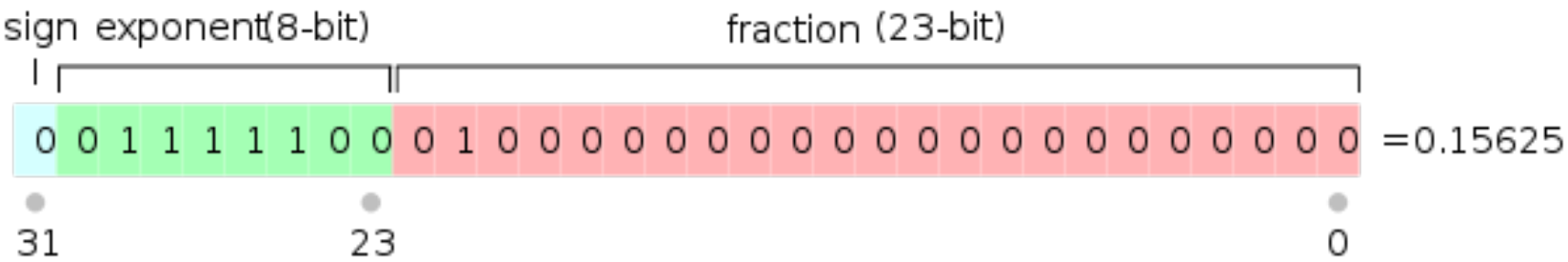
Standard IEEE 754: Calcolo e rappresentazione della mantissa.

Esempio

Calcolo della mantissa o significando in formato **IEEE 754** a singola precisione (23 bit) per il numero $-\mathbf{113.25_{10}}$

3. Ricordando che **la cifra alla sinistra delle virgola non si rappresenta**, la mantissa (M) sarà costituita dai segg. 23 bit:
   - Bit **22-15**: 11000101
   - Bit **14-0**: 00000000000000 (padding)
4. Quindi sarà $M = 11000101000000000000000$

Standard IEEE 754: Calcolo e rappresentazione dello esponente

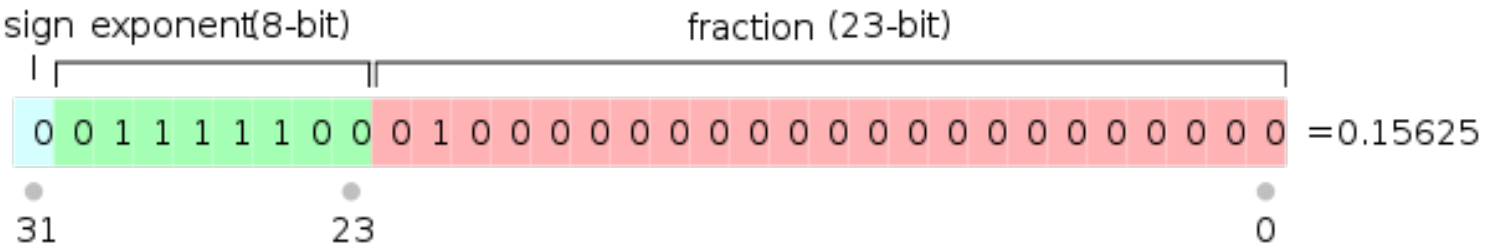


Con 8 bit, in teoria si avrebbero 256 combinazioni.

In pratica:

- I valori 0 e 255 sono **riservati per usi speciali** (se ne parlerà dopo).
- In pratica **si rappresentano 254 valori**, da -126 a $+127$.

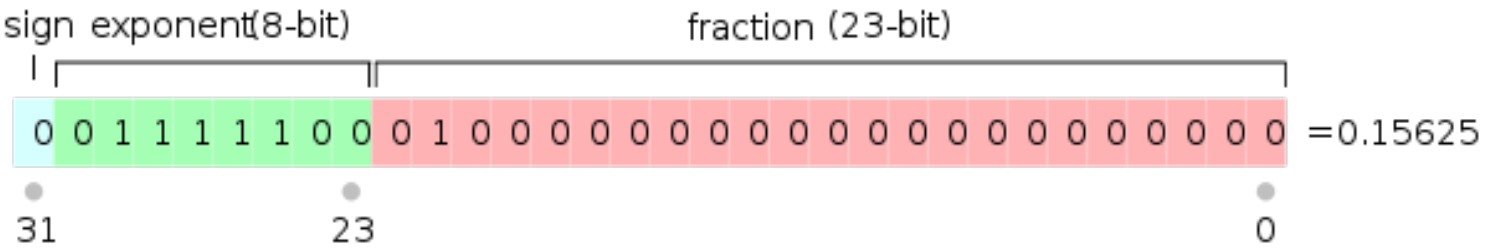
Standard IEEE 754: Calcolo e rappresentazione dello esponente



Infine, il valore dello esponente si rappresenta sommando un valore k detto **bias**:

- $\mathbf{E} = \mathbf{e} + \mathbf{k}$.
- **E** è il valore **effettivamente rappresentato** nel campo esponente.
- **k** è il **bias**.
- **e** è il valore **esponente** ottenuto durante il **calcolo della mantissa**.

Standard IEEE 754: Calcolo e rappresentazione dello esponente



Nel caso dei floating point a 32 bit si ha $\mathbf{k} = +\mathbf{127}$, e valori rappresentabili $-126 \leq e \leq +127$.

Dunque se $\mathbf{E} = \mathbf{e} + \mathbf{k}$ allora $\mathbf{1} \leq \mathbf{E} \leq \mathbf{254}$

In tal modo:

- (+) Le codifiche 0 e 255 si possono riservare per usi speciali.
- (+) Non si deve rappresentare il segno per il campo esponente (E non ha segno).

Standard IEEE 754: Calcolo e rappresentazione dello esponente

Nello ESEMPIO precedente era

$$113.25_{10} = 1110001.01_2 = \mathbf{1.11000101} \times \mathbf{2^6}.$$

Quindi $\mathbf{e = 6}$.

Allora $\mathbf{E = 6 + k = 6 + 127 = 133 = 10000101}$.

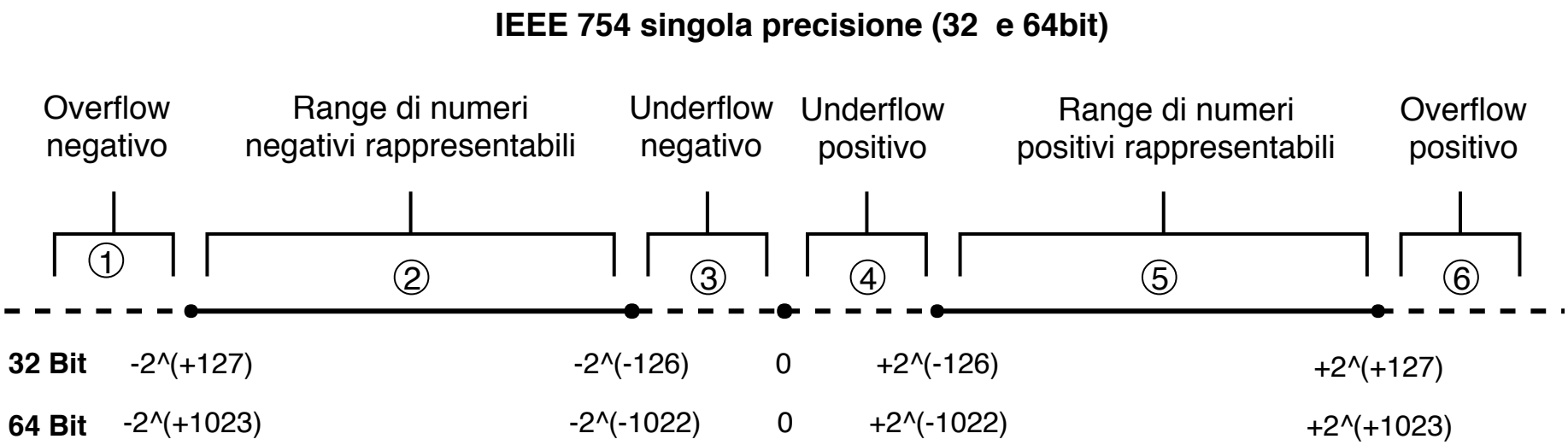
Infine, il **segno**: dato che il numero -113.25 è negativo, $\mathbf{s{=}1}$.

Quindi la codifica a 32 bit floating point IEEE 754 del numero -113.25 è la seguente:

s	E	M
1	**10000101**	**11000101000000000000000**

Standard IEEE 754: intervalli numerici

NB: Gli intervalli 2 e 5 sono costituiti da **un numero finito di elementi** che costituiscono un sottoinsieme di $\mathbb{R}$.

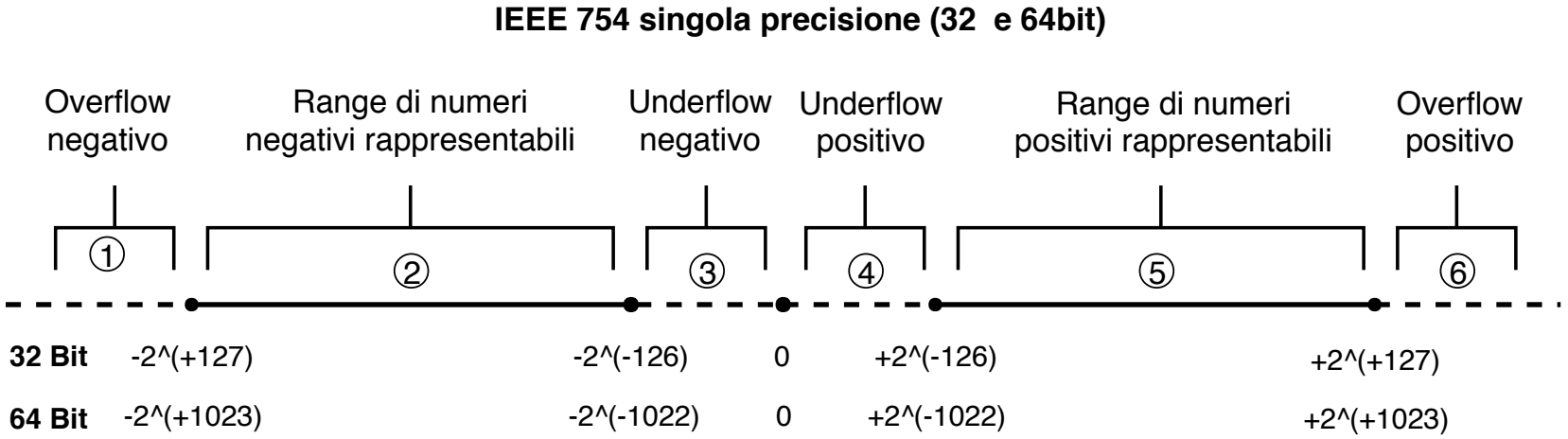


Standard IEEE 754: approssimazioni

1) **Valore non rappresentabile con un numero finito di cifre**, la sua rappresentazione sarà **troncata**.

ESEMPIO:$4.35_{10} = 100.010\overline{1100}$.

Il numero 4.35 ricade all'interno del range dell'intervallo 5, ma non ne fa parte (non è rappresentabile senza approssimazione).



Standard IEEE 754: approssimazioni

2) Valore con un **numero di cifre significative** maggiore del numero massimo rappresentabile nel campo mantissa (o significando).

Esempio

Si consideri il numero **9876543.25**.
Codifica floating point IEEE 754 a singola precisione (32 bit):

$$\begin{aligned} \mathbf{9876543.25_{10}} &= 100101101011010000111111.01 = \\ &= \mathbf{1.00101101011010000111111}01 \times 2^{23}. \end{aligned}$$

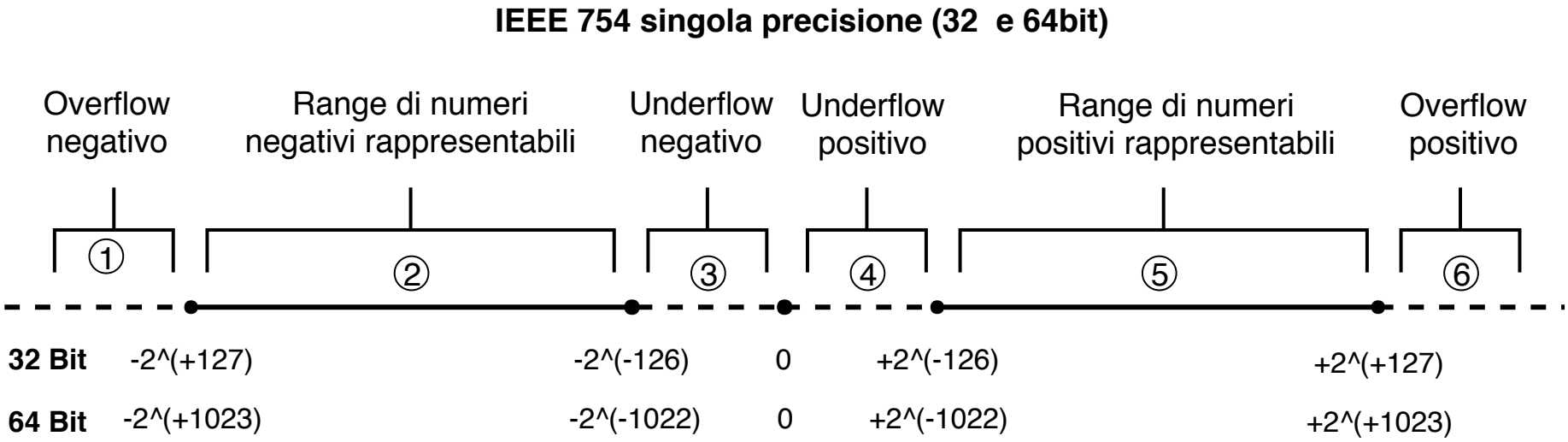
NB: Mantissa (o significando) di lunghezza **25**$>$ 23

Ma per **floating point 32 bit** lunghezza massima mantissa **23 bit!**.

$\Longrightarrow$ il valore sarà memorizzato in modo approssimato!

Standard IEEE 754: approssimazioni

Osservazione: Il numero **9876543.25** "cade" nell'intervallo 5 di IEEE 754 a singola precisione.



Standard IEEE 754: approssimazioni

Precisione di una rappresentazione in virgola mobile

In generale una certa rappresentazione floating point è caratterizzata da **una precisione p**.

La precisione p è costituita dal numero di **cifre significative** che è possibile rappresentare in quel determinato formato.

Esempio

Si consideri il numero $\mathbf{1234.030405887000}_{10}$.

Le cifre significative sono costituite dalla sequenza **1234030405887**

Homework H7.6

Calcolare segno, esponente e mantissa secondo IEEE 754 per i seguenti numeri reali.

A) -754.1234567

B) 314.6543

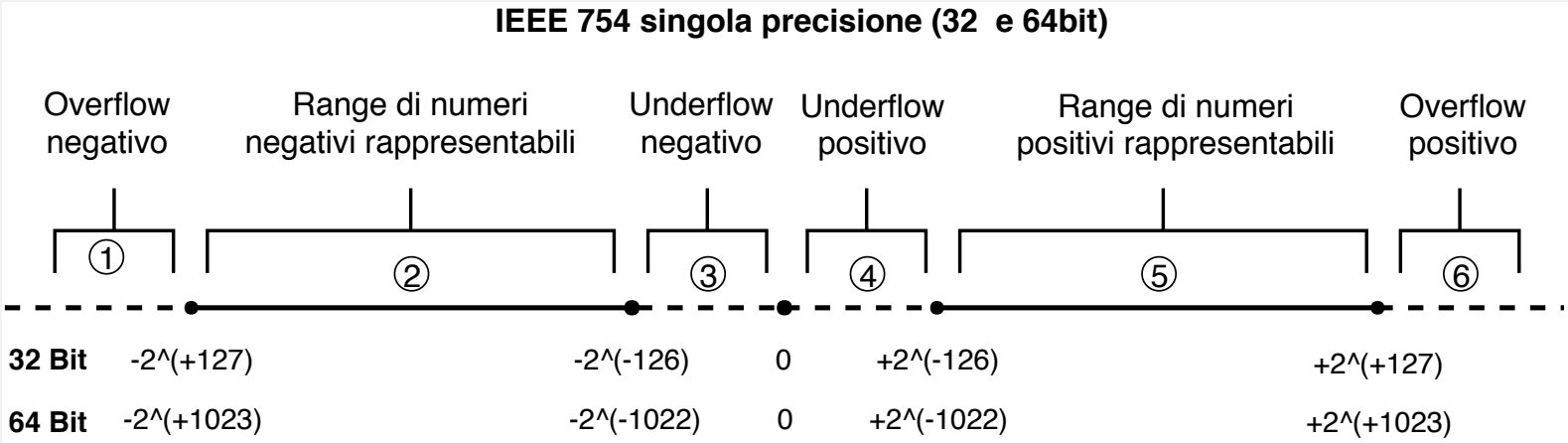
C) 112.659834321

Standard IEEE 754: Overflow

Overflow: Il risultato numerico di una certa espressione aritmetica è **maggiore**, in valore assoluto, del valore massimo rappresentabile.

Una situazione di overflow viene rilevata nel campo esponente!

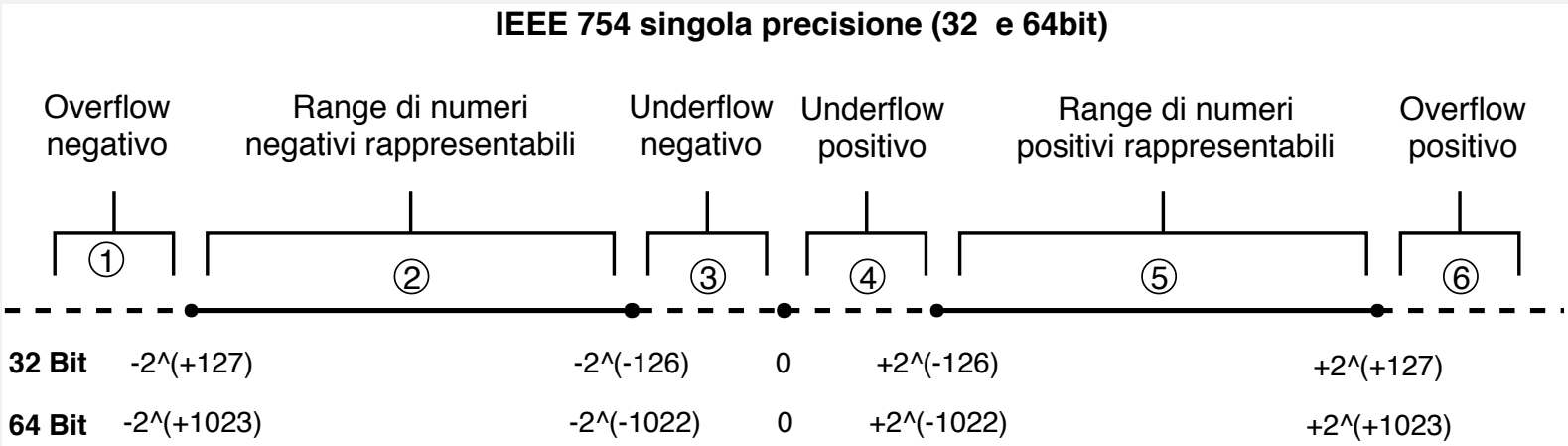
Overflow per numeri floating point



Standard IEEE 754: Underflow

Underflow: Il risultato di una operazione è **minore**, in valore assoluto, del più piccolo valore rappresentabile.

Una situazione di underflow viene rilevata nel campo esponente!



IEEE 754: valori speciali

IEEE 754 riserva alcune combinazioni di bit per **valori speciali**.

Valori Speciali IEEE 754

$\pm$**Inf**: Divisione di numeri in virgola mobile per zero oppure casi di **overflow (positivo o negativo)**: $\pm\frac{x}{0}$.

NaN Forme indeterminate (risultato indefinito) : $\frac{0}{0}, \frac{\pm Inf}{\pm Inf}$, oppure ancora $+Inf - Inf$, e cosi via.

NB: $\pm$**Inf** e **NaN** sono rappresentati con alcune combinazioni di bit non usati nella rappresentazione dello esponente (0 e 255, vedi slide precedenti..)

$\pm$**0** Underflow codificati come **zero con segno**, a seconda che il risultato della espressione sia positivo o negativo.

Rappresentazione dei numeri nei calcolatori

Per rappresentare i numeri nei **calcolatori** si usa la **rappresentazione base 2**.

Questa differisce a seconda che si debbano rappresentare **numeri relativi** (dell'insieme $\mathbb{Z}$) o numeri **reali** (dell'insieme $\mathbb{R}$).

Riassumendo..

Rappresentazione di numeri interi

- Un bit (opzionale) per il **segno**
- **Tutti i numeri all'interno del range specificato** dalla rappresentazione **sono rappresentabili** (NO approssimazione, NO underflow).
- Possibile **OVERFLOW** (negativo o positivo)
- Al fine di **non sprecare spazio in memoria** e **non incorrere in overflow**, è importante scegliere una opportuna rappresentazione
  1. numero di bit (ES: 16 o 32);
  2. eventuale presenza del bit per il segno.

Riassumendo..

Numeri in virgola mobile ("floating point")

- Standard IEEE 754 per singola precisione (32 bit) o doppia precisione (64 bit).
- Codifica/rappresentazione *compatta* mediante **significando** (o mantissa), **segno** ed **esponente**.
- La **precisione** p di una codifica floating point è rappresentata dal **numero di cifre significative** che è possibile rappresentare nella mantissa o significando.
  - 6 cifre per la precisione singola
  - 15 cifre per la precisione doppia

Riassumendo..

Numeri in virgola mobile ("floating point")

- **Errori di approssimazione** quando
  - il numero non è rappresentabile con numero finito di cifre.
  - il numero di cifre significative del numero da rappresentare è maggiore della precisione p
- **Underflow** quando il numero da rappresentare, in valore assoluto, è troppo piccolo per essere rappresentato.
- **Overflow**: il valor assoluto del numero è maggiore, del numero piu grande (in valore assoluto) rappresentabile con quella codifica.

NOTE

La teoria inerente la codifica o rappresentazione delle informazioni sarà trattata con il dovuto rigore durante il corso di Fondamenti di Programmazione (Barbanera/Madonia).

http://www.dmi.unict.it/barba/FONDAMENTI/PROGRAMMI-TESTI/READING-MATERIAL/TwoComplement/twoComplement.htm

La codifica delle informazioni al calcolatore viene trattata anche nel corso di Architetture degli Elaboratori (Prof. Santoro).