Tecniche della Programmazione, lez. 11 seconda parte

Sistemi di numerazione e aritmetica binaria

Rappresentazione dei numeri

- interi: complemento a due
- caratteri
- reali: Floating Point

E parliamo anche di `espressione condizionale` e *obfuscation* …

Remember

con N cifre binarie (N bit) si possono scrivere i numeri naturali da 0 a 2^N-1

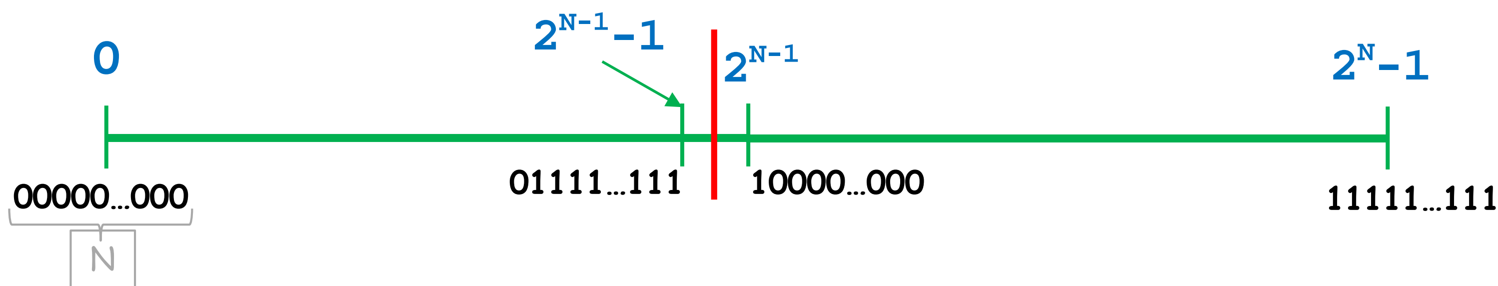
intervallo di rappresentabilita` per i numeri naturali espressi in forma binaria, con N cifre (N bit):

$$[0, 2^N-1]$$

Remember

intervallo di rappresentabilita` per i numeri naturali espressi in forma binaria, con N cifre (N bit):

$$[0, 2^N-1]$$



ESEMPIO: Numerali con 4 bit

0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	10
1 0 1 1	11
1 1 0 0	12
1 1 0 1	13
1 1 1 0	14
1 1 1 1	15

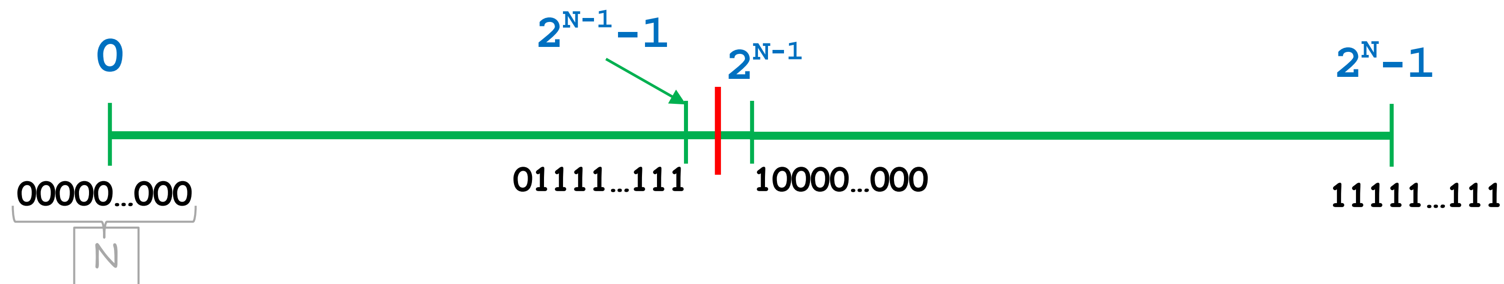
con N = 4, abbiamo 2^4 numerali, cioè 16 da 0000 a 1111 che possono rappresentare i numeri naturali da 0 a 2^4-1 cioè l'intervallo [0, 15]

☺ replicare questa slide, intera, per N=5

$[0, 2^5-1] = [0, 32-1] = [0, 31]$

Remember

con N cifre binarie (N bit) si possono scrivere i numeri naturali da 0 a 2^N-1



Se N = 32, allora abbiamo 2^{32} numerali,

da 00000000000000000000000000000000 a 11111111111111111111111111111111

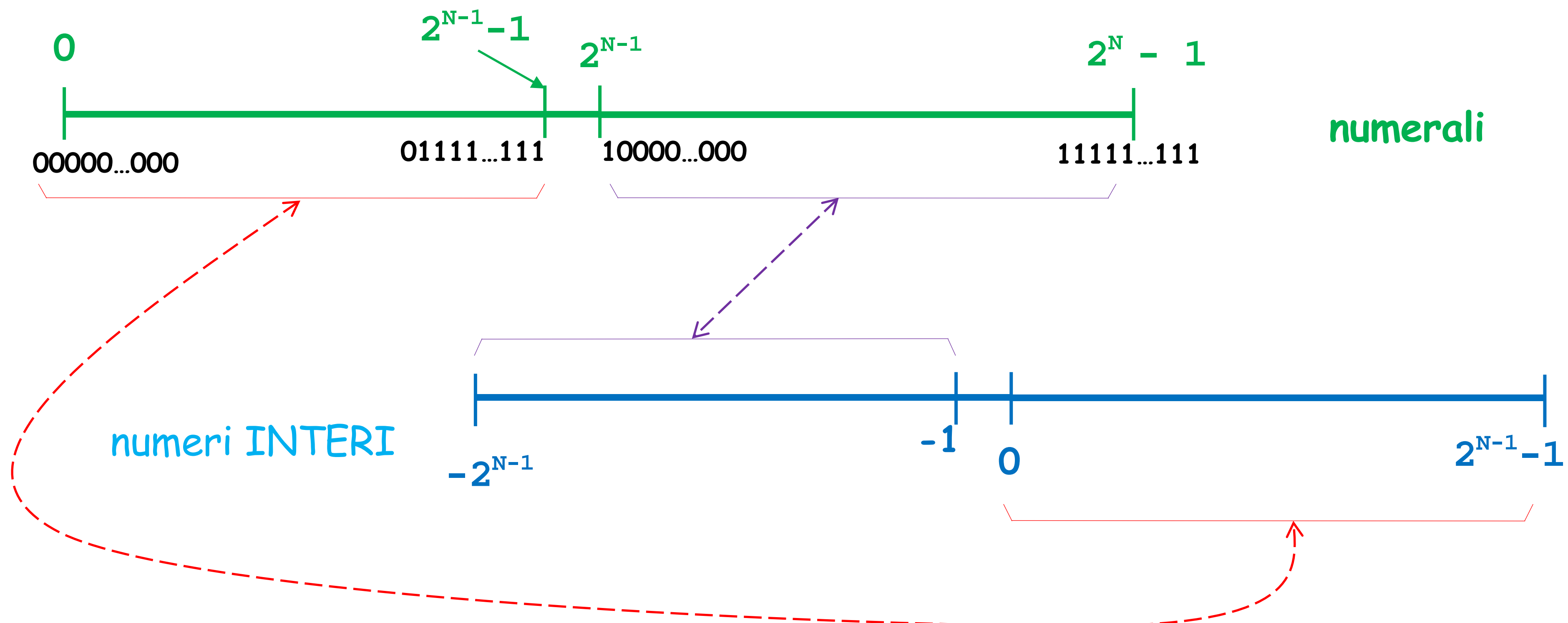
che possono rappresentare i numeri naturali
da 0 a 2^{32}-1
cioè l'intervallo $[0, 2^{32}-1]$

Rappresentazione dei numeri interi in *Complemento a 2*

Solo che non si può vivere solo con i naturali …
Vorremmo rappresentare, **con i numerali disponibili**, quanti più numeri INTERI possibile, un pò positivi e un pò negativi (diciamo metà e metà)

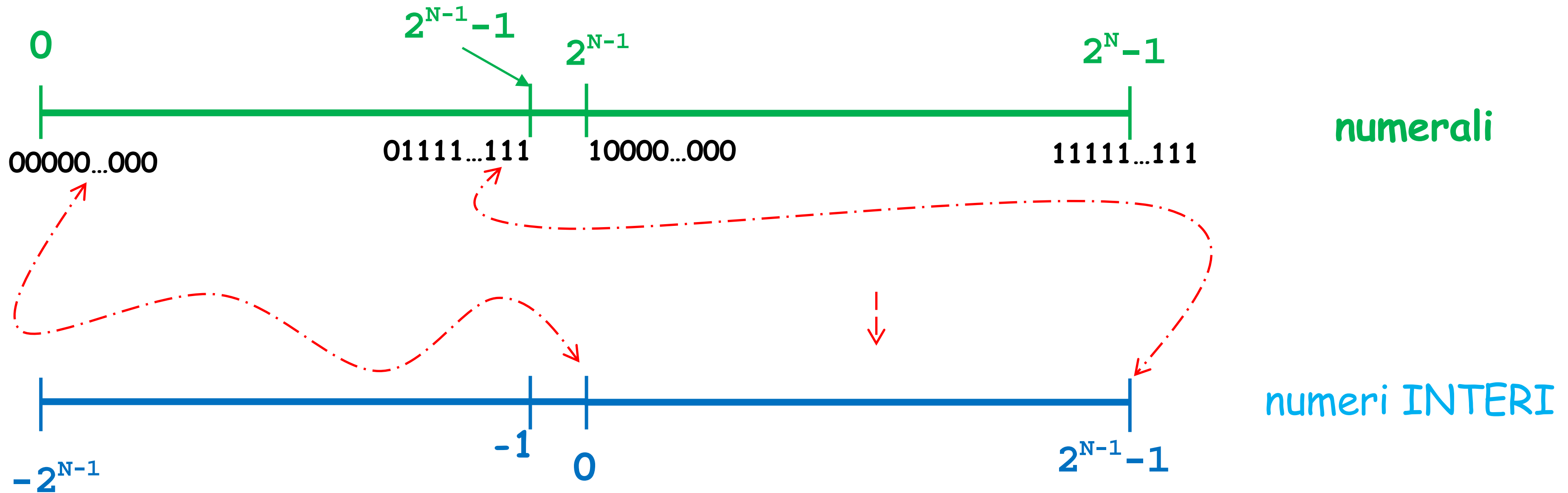
Con N cifre binarie (N bit) i **numerali disponibili** sono 2^N :

- **ne usiamo metà per i numeri negativi (sono 2^{N-1} numerali)**
- **usiamo gli altri 2^{N-1} numerali per i numeri positivi e per lo zero**



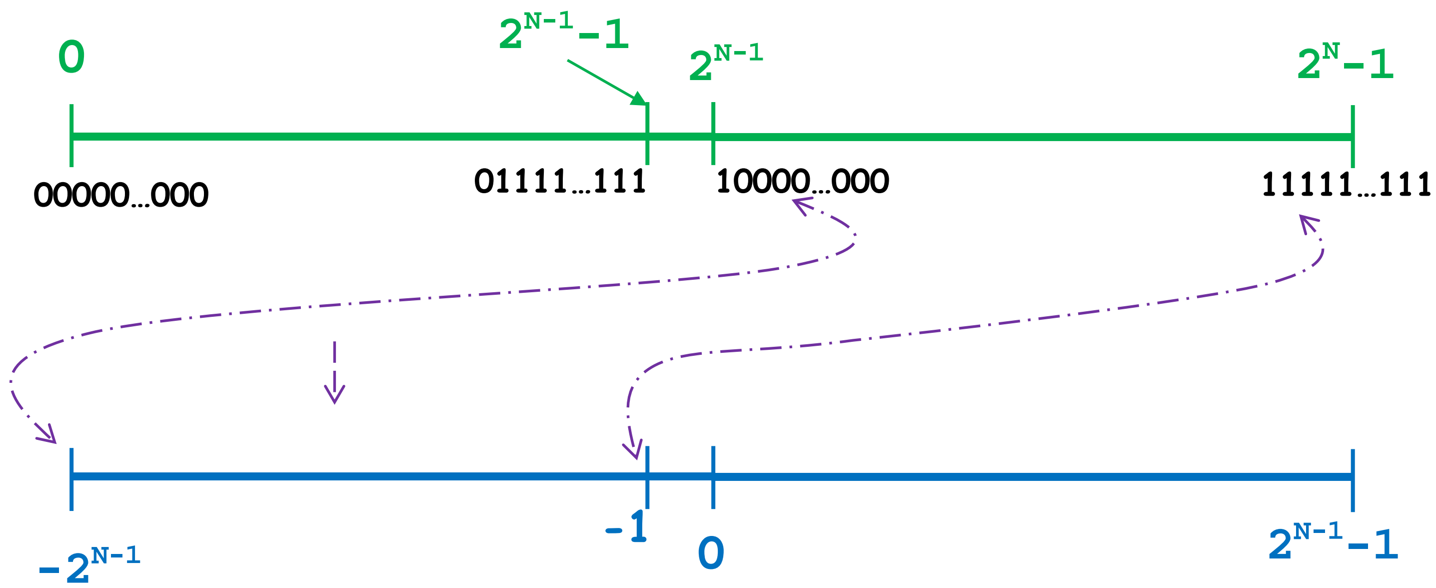
Rappresentazione in *Complemento a 2 (Tipo `int`)*

dettaglio su alcuni numeri / numerali estremi ...



Rappresentazione in *Complemento a 2 (Tipo `int`)*

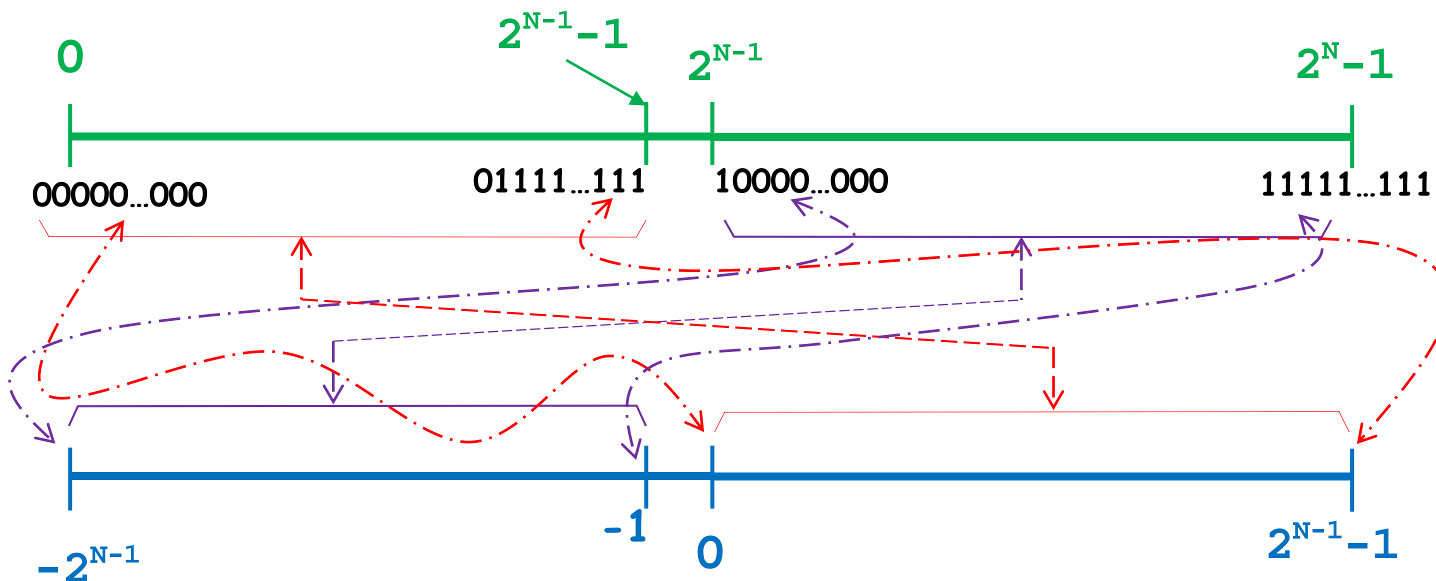
dettaglio su alcuni numeri / numerali estremi ...



Rappresentazione in *Complemento a 2* (*Tipo `int`*)

Solo che non si può vivere solo con i naturali ...
Vorremmo rappresentare, con i numerali disponibili, quanti più numeri INTERI possibile, un pò positivi e un pò negativi (diciamo metà e metà)

Con N cifre binarie (N bit) i **numerali disponibili** sono 2^N :
- **ne usiamo metà per i numeri negativi (sono 2^{N-1} numerali)**
- **usiamo gli altri 2^{N-1} numerali per i numeri positivi e per lo zero**



Con N = 32, i 2^{32} numerali rappresentano i numeri interi nell'intervallo $[-2^{31}, 2^{31}-1]$ (intervallo di rappresentabilità)

Rappresentazione in *Complemento a 2* (Tipo `int`)

Solo che non si può vivere solo con i naturali ...
Vorremmo rappresentare, con i numerali disponibili, quanti più numeri INTERI possibile, un pò positivi e un pò negativi (diciamo metà e metà)

Con N cifre binarie (N bit) i **numerali disponibili** sono 2^N :
- **ne usiamo metà per i numeri negativi (sono 2^{N-1} numerali)**
- **usiamo gli altri 2^{N-1} numerali per i numeri positivi e per lo zero**

Con N = 32, i 2^{32} numerali rappresentano i numeri interi nell'intervallo $[-2^{31}, 2^{31}-1]$ (intervallo di rappresentabilità)

NB
Se N e` il numero di bit usati per i numerali, l'intervallo di rappresentabilita` dei numeri interi in complemento a 2 si esprime come

$$[-2^{N-1}, 2^{N-1}-1]$$

Tipo int (rappresentazione in Complemento a 2)

Solo che non si può vivere solo con i naturali ...
Vorremmo rappresentare, con i numerali disponibili, quanti più numeri INTERI possibile, un pò positivi e un pò negativi (diciamo metà e metà)

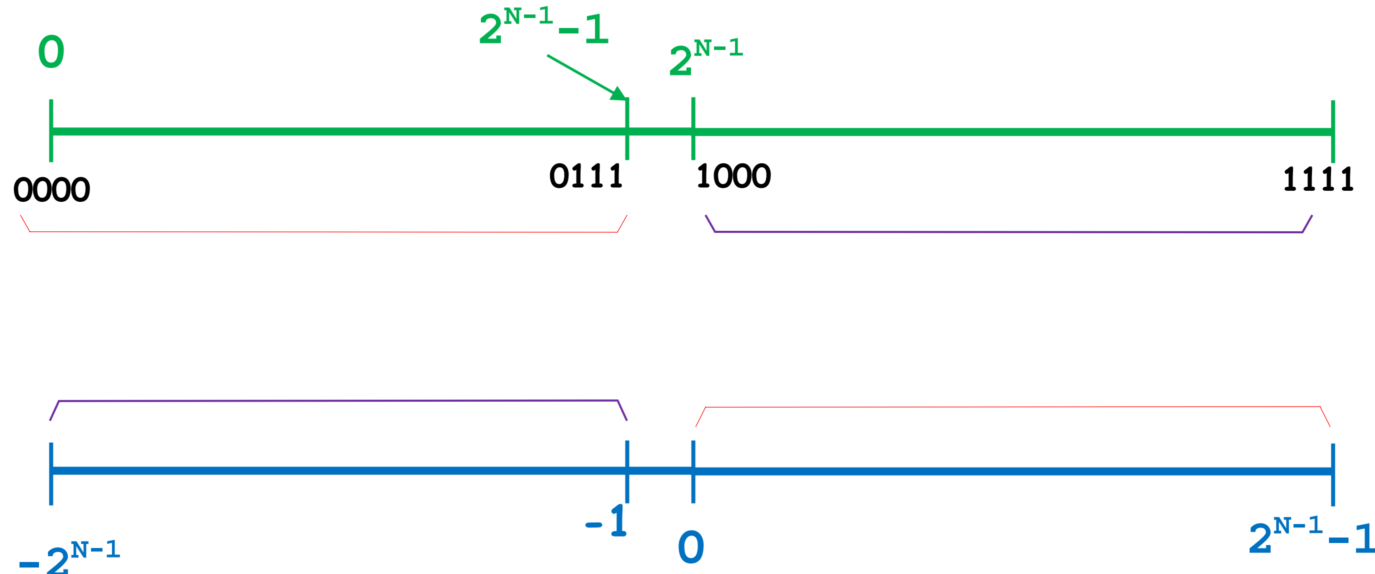
Con N cifre binarie (N bit) i numerali disponibili sono 2^N :

- **ne usiamo metà per i numeri negativi (sono 2^{N-1} numerali)**
- **usiamo gli altri 2^{N-1} numerali per i numeri positivi e per lo zero**

ESEMPIO: Numerali con 4 bit

	naturali	interi
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	3	3
0 1 0 0	4	4
0 1 0 1	5	5
0 1 1 0	6	6
0 1 1 1	7	7
1 0 0 0	8	-8
1 0 0 1	9	-7
1 0 1 0	10	-6
1 0 1 1	11	-5
1 1 0 0	12	-4
1 1 0 1	13	-3
1 1 1 0	14	-2
1 1 1 1	15	-1

Con N = 4, i 24 numerali rappresentano i numeri interi nell'intervallo $[-2^3 , 2^3-1] = [-8, 7]$

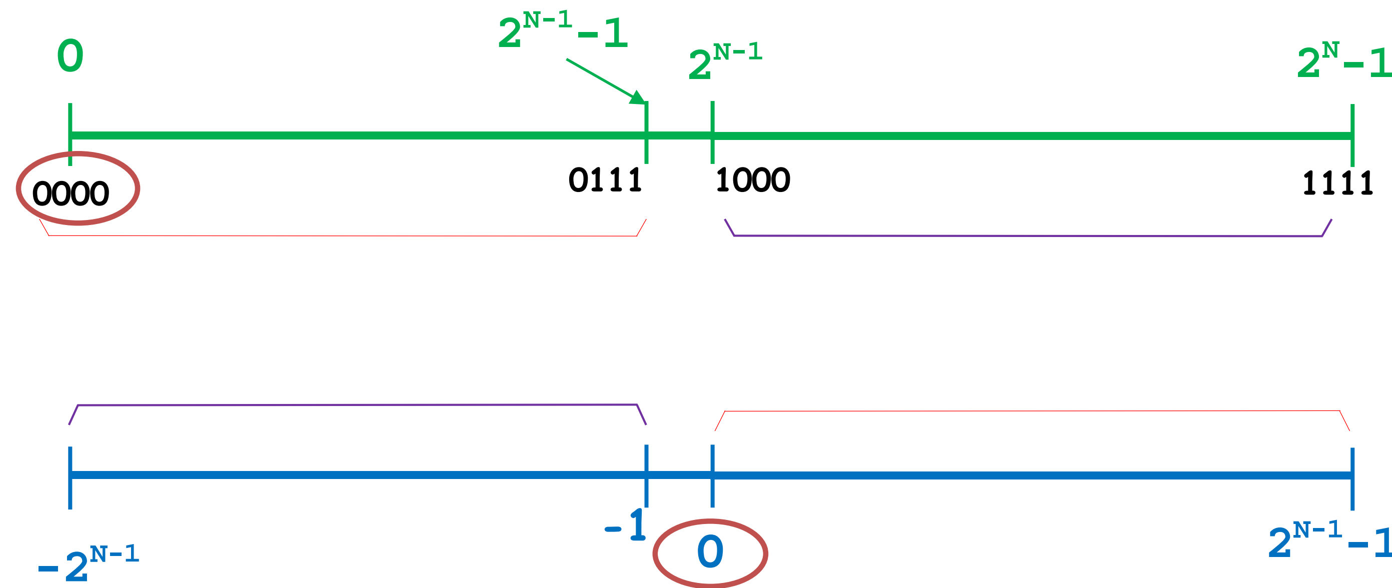


Tipo int (rappresentazione in Complemento a 2)

Solo che non si può vivere solo con i naturali ...
Vorremmo rappresentare, con i numerali disponibili, quanti più numeri INTERI possibile, un pò positivi e un pò negativi (diciamo metà e metà)

Con N cifre binarie (N bit) i **numerali disponibili** sono 2^N :

- **ne usiamo metà per i numeri negativi (sono 2^{N-1} numerali)**
- **usiamo gli altri 2^{N-1} numerali per i numeri positivi e per lo zero**



ESEMPIO: Numerali con 4 bit

	naturali	interi
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

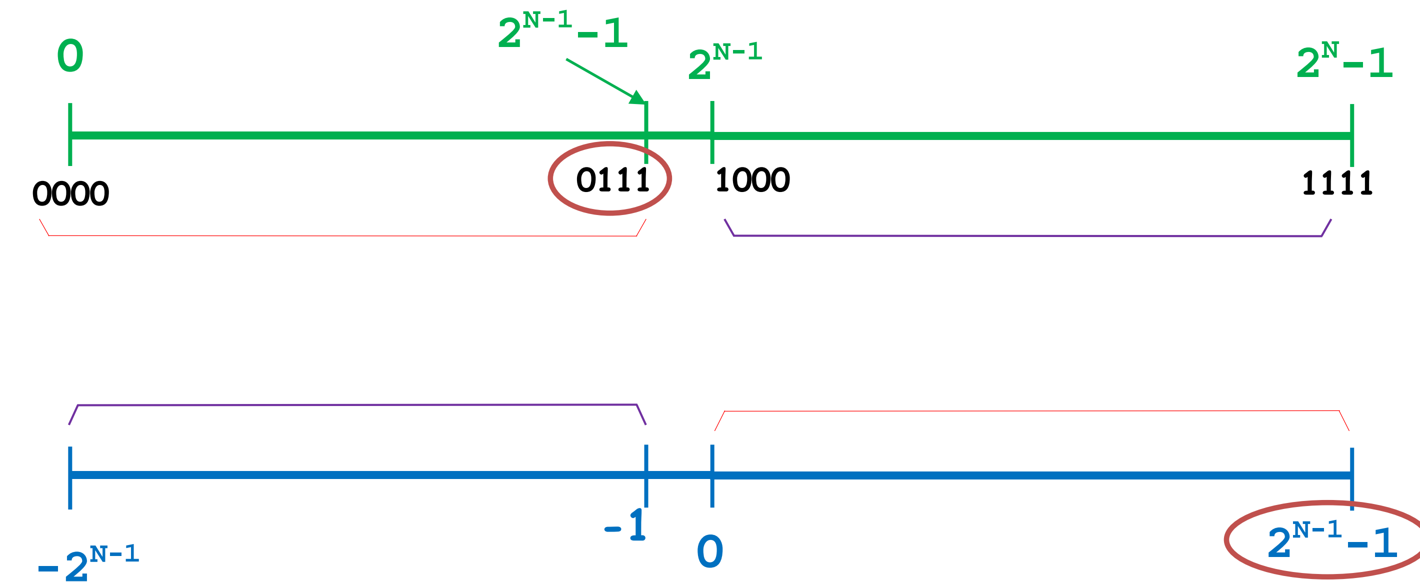
Con N = 4, i 2^4 numerali rappresentano i numeri interi nell'intervallo $[-2^3 , 2^3-1]$

Tipo int (rappresentazione in Complemento a 2)

Solo che non si può vivere solo con i naturali …
Vorremmo rappresentare, con i numerali disponibili, quanti più numeri INTERI possibile, un pò positivi e un pò negativi (diciamo metà e metà)

Con N cifre binarie (N bit) i **numerali disponibili** sono 2^N :

- **ne usiamo metà per i numeri negativi (sono 2^{N-1} numerali)**
- **usiamo gli altri 2^{N-1} numerali per i numeri positivi e per lo zero**



Con N = 4, i 2^4 numerali rappresentano i numeri interi nell'intervallo $[-2^3-, 2^3-1]$

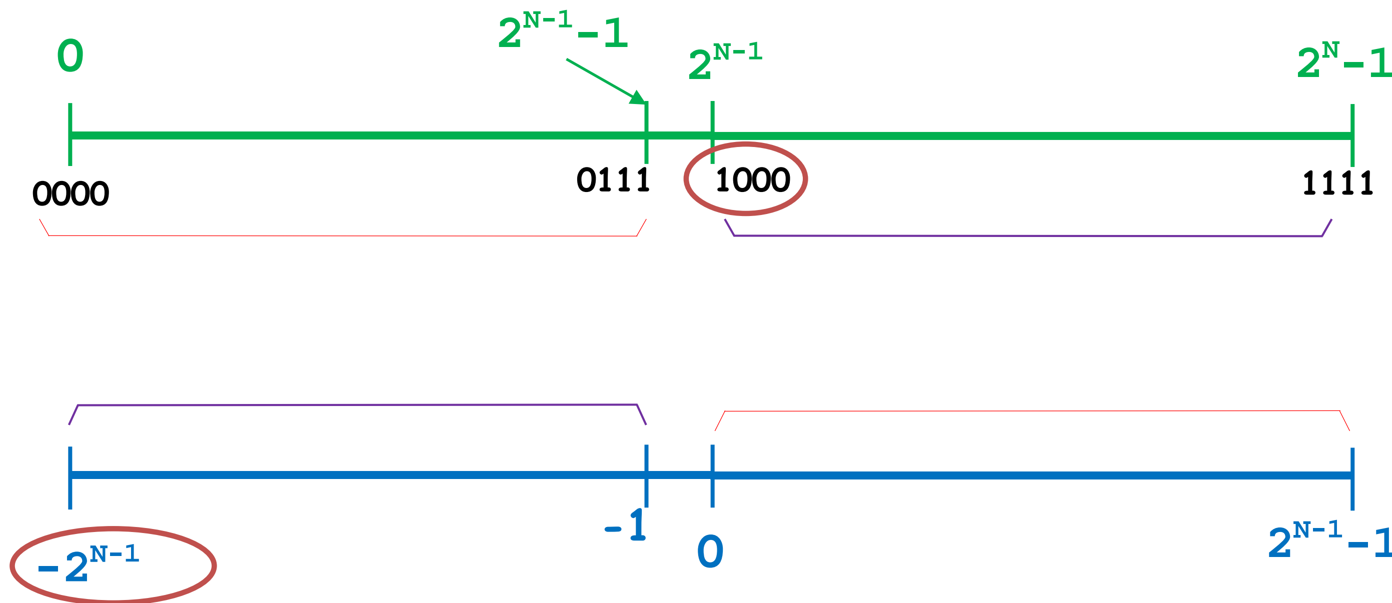
ESEMPIO: Numerali con 4 bit

	naturali	interi
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Tipo int (rappresentazione in Complemento a 2)

Solo che non si può vivere solo con i naturali ...
Vorremmo rappresentare, con i numerali disponibili, quanti più numeri INTERI possibile, un pò positivi e un pò negativi (diciamo metà e metà)

Con N cifre binarie (N bit) i **numerali disponibili** sono 2^N :
- **ne usiamo metà per i numeri negativi (sono 2^{N-1} numerali)**
- **usiamo gli altri 2^{N-1} numerali per i numeri positivi e per lo zero**



ESEMPIO: Numerali con 4 bit

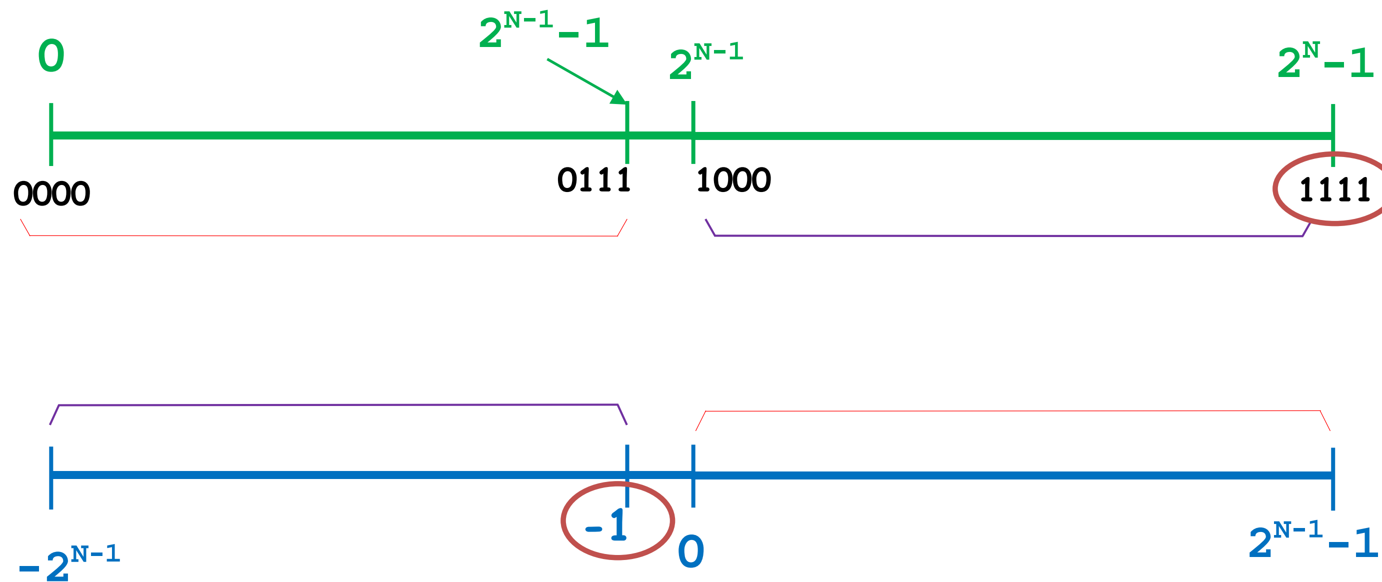
	naturali	interi
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Con N = 4, i 2^4 numerali rappresentano i numeri interi nell'intervallo $[-2^3-, 2^3-1]$

Tipo int (rappresentazione in Complemento a 2)

Solo che non si può vivere solo con i naturali ...
Vorremmo rappresentare, con i numerali disponibili, quanti più numeri INTERI possibile, un pò positivi e un pò negativi (diciamo metà e metà)

Con N cifre binarie (N bit) i **numerali disponibili** sono 2^N :
- **ne usiamo metà per i numeri negativi (sono 2^{N-1} numerali)**
- **usiamo gli altri 2^{N-1} numerali per i numeri positivi e per lo zero**



ESEMPIO: Numerali con 4 bit

	naturali	interi
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Con N = 4, i 2^4 numerali rappresentano i numeri interi nell'intervallo $[-2^3-, 2^3-1]$

Tipo int: circolarità della rappresentazione

0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	3	3
0 1 0 0	4	4
0 1 0 1	5	5
0 1 1 0	6	6
0 1 1 1	7	7
1 0 0 0	8	-8
1 0 0 1	9	-7
1 0 1 0	10	-6
1 0 1 1	11	-5
1 1 0 0	12	-4
1 1 0 1	13	-3
1 1 1 0	14	-2
1 1 1 1	15	-1

Rappresentazione in Complemento a 2 (calcolo)

Dato un numero **n**, lo rappresentiamo in complemento a 2 tramite il numerale binario **rappr(n)**

Attention please – ci sono tre momenti in questo ragionamento …

1) Con quanti bit, al minimo, possiamo scrivere **rappr(n)** ?

   Sia $\overline{\mathbf{N}}$ questo numero...

2) Stabilito $\overline{\mathbf{N}}$, decidiamo il numero **N** con cui vogliamo scrivere **rappr(n)**

   Sara` **N >=** $\overline{\mathbf{N}}$ e di solito scegliamo **N =** $\overline{\mathbf{N}}$ **...**

3) e poi calcoliamo il numerale **rappr(n)**, scritto su N bit

Rappresentazione in Complemento a 2 (fasi 1 e 2)

Dato un numero **n**, lo rappresentiamo in complemento a 2 tramite il numerale binario **rappr(n)**

Attention please – ci sono tre momenti in questo ragionamento …

1) "Qual è il minimo numero -- $\overline{\mathbf{N}}$ -- di bit, che basta per rappresentare n ?"

La determinazione del numero $\overline{\mathbf{N}}$ "minimo necessario" proviene da un'analisi dell'intervallo di rappresentabilità

per diversi K si considera l'intervallo di rappresentabilita`

$[-2^{k-1}, 2^{k-1}-1]$

facendosi queste domande
... il numero n è nell'intervallo di rappresentabilità per numerali di K bit?
... K è il minimo numero di bit necessari per rappresentare il numero n?

2) Facciamo che il numero di bit con cui scriviamo **rappr(n)** sia $N = \overline{N}$ ... per brevita`

Rappresentazione in Complemento a 2 (fase 3: calcolo)

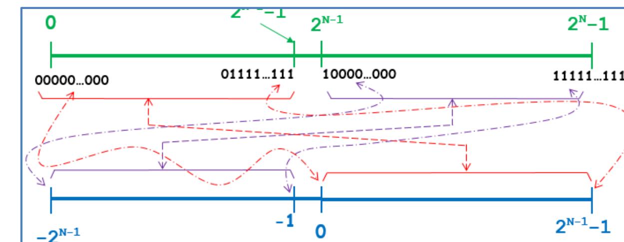
Dato un numero **n**, lo rappresentiamo in complemento a 2 tramite il numerale binario **rappr(n)**

Attention please – ci sono tre momenti in questo ragionamento …

3) Con N il numero di bit usati per scrivere i numerali

- se n è positivo && n <= $2^{N-1}-1$ **$n \in [0, 2^{N-1}-1]$**
  rappr(n) = il normale numerale binario per n, in N bit
  (il primo bit e\` 0, ovviamente)

- se n è negativo && n >= -2^{N-1} **$n \in [-2^{N-1}, 0)$**
  rappr(n) = il numerale binario, in N bit, che rappresenta il numero positivo $2^N - |n|$
  (com'è il primo bit?)



esempio: N=12

rappr(796) = "796 in binario su 12 bit" = ☺
rappr(-796) = "rappr(2^{12} – |-796|) in 12 bit" = ☺

Rappresentazione in Complemento a 2 (fase 3: calcolo)

3) Con N il numero di bit usati per scrivere i numerali

- se n è positivo && n <= 2^{N-1} -1 — **n ∈ [0, 2^{N-1} -1]**

  rappr(n) = il normale numerale binario per n, in N bit
  (il primo bit e\` 0, ovviamente)

- se n è negativo && n >= -2^{N-1} — **n ∈ [-2^{N-1}, 0)**

  rappr(n) = il numerale binario, in N bit, che rappresenta il numero positivo $2^N - |n|$
  (com'è il primo bit? E\` 1, ovviamente)

esempio: N=12

rappr(796) = "796 in binario su 12 bit" =
rappr(-796) = "rappr(2^N – |n|) in N bit" =

le due rappresentazioni sono tra le le seguenti? Quale è quale?

110011100100 001100011100
001100011100 110011100100
110011100100 001100011100

Rappresentazione in Complemento a 2 (fase 3: calcolo)

3) Con N il numero di bit usati per scrivere i numerali

- se n è positivo && $n <= 2^{N-1} - 1$ — $n \in [0, 2^{N-1} - 1]$

  rappr(n) = il normale numerale binario per n, in N bit

  (il primo bit e` 0, ovviamente)

- se n è negativo && $n >= -2^{N-1}$ — $n \in [-2^{N-1}, 0)$

  rappr(n) = il numerale binario, in N bit, che rappresenta il numero positivo $2^N - |n|$

  (com'è il primo bit? E` 1, ovviamente)

esempio: N=12

rappr(796) = "796 in binario su 12 bit" = 001100011100

(numero naturale 796)

rappr(-796) = "rappr($2^N - |n|$) in N bit"

= 4096-796 in binario su 12 bit = 110011100100

(numero naturale 3300)

Rappresentazione in Complemento a 2 (esempi)

intervallo dei numeri rappresentabili in compl. a 2 con N bit (2^N numerali)

$$[-2^{(N-1)} , 2^{(N-1)} - 1]$$

con 4 bit, ci sono 2alla4 numerali disponibili:
intervallo di rappresentabilità $[-2^3 , 2^3 - 1] = [-8, 7]$

intervallo dei numeri rappresentabili in compl. a 2 con 8 bit (2^8 numerali)

$$[-2^7 , 2^7 - 1] = [-128, 127]$$

adesso per esempio assumiamo n = 796

… con 10 bit (2^{10} numerali) ☺ intervallo di rappresentabiltà?
e … n è nell'intervallo?

Rappresentazione in Complemento a 2 (esempi)

intervallo dei numeri rappresentabili in compl. a 2 con N bit (2^N numerali)

$$[-2^{(N-1)} , 2^{(N-1)} - 1]$$

con 4 bit, ci sono 2alla4 numerali disponibili:
intervallo di rappresentabilità $[-2^3 , 2^3 - 1] = [-8, 7]$

intervallo dei numeri rappresentabili in compl. a 2 con 8 bit (2^N numerali)

$$[-2^7 , 2^7 - 1] = [-128, 127]$$

adesso per esempio assumiamo n = 796

… con 10 bit (2^{10} numerali) $[-2^9 , 2^9 - 1]$ = ☺

Rappresentazione in Complemento a 2 (esempi)

...
intervallo dei numeri rappresentabili in compl. a 2 con 8 bit (2^N numerali)

$[-2^7 , 2^7 - 1] = [-128, 127]$

adesso per esempio assumiamo n = 796

con 10 bit (2^{10} numerali) $[-2^9 , 2^9 - 1] = [-512, 511]$

no, 796 non è nell'intervallo
10 bit non bastano

Rappresentazione in Complemento a 2 (esempi)

...
intervallo dei numeri rappresentabili in compl. a 2 con 8 bit (2^N numerali)
$[-2^7 , 2^7 - 1] = [-128, 127]$

adesso per esempio assumiamo n = 796

... con 10 bit (2^{10} numerali) $[-2^9 , 2^9 - 1] = [-512, 511]$

... con 11 bit (2^{11} numerali) ☺

Rappresentazione in Complemento a 2 (esempi)

...
intervallo dei numeri rappresentabili in compl. a 2 con 8 bit (2^N numerali)

$[-2^7 , 2^7 - 1] = [-128, 127]$

adesso per esempio assumiamo n = 796

... con 10 bit (2^{10} numerali) $[-2^9 , 2^9 - 1] = [-512, 511]$

... con 11 bit (2^{11} numerali) $[-2^{10} , 2^{10} - 1] = [-1024, 1023]$

796 è nell'intervallo (11 bit bastano)

remember: $N >= \overline{N}$

... con 12 bit (2^{12} numerali) ☺

... con 13 bit (2^{13} numerali) ☺

Rappresentazione in Complemento a 2 (esempi)

intervallo dei numeri rappresentabili in compl. a 2 con 8 bit (2^N numerali)

$[-2^7 , 2^7 - 1] = [-128, 127]$

... con 10 bit (2^{10} numerali) $[-2^9 , 2^9 - 1]$ = $[-512, 511]$

$[-1024, 1023]$

... con 11 bit (2^{11} numerali) $[-2^{10} , 2^{10} - 1]$ = ☺

796 è nell'intervallo

... con 12 bit (2^{12} numerali) $[-2^{11} , 2^{11} - 1]$ = $[-2048, 2047]$

796 è nell'intervallo

... con 13 bit (2^{13} numerali) $[-4096, 4095]$

796 è nell'intervallo

Rappresentazione in Complemento a 2 (esempio)

(N=numero di bit usati per scrivere i numerali)

Dato un numero n,

- se n è positivo && $n <= 2^{N-1} -1$

  rappr(n) = normale numerale binario in N bit (il primo bit è necessariamente 0)

- se n è negativo && $n >= -2^{N-1}$

  rappr(n) = $2^N - |n|$

esempio:

-796 quanti bit servono al minimo? 10?

Rappresentazione in Complemento a 2 (esempio)

(N=numero di bit usati per scrivere i numerali)

Dato un numero n,

- se n è positivo && $n <= 2^{N-1} - 1$
  rappr(n) = normale numerale binario in N bit (il primo bit è necessariamente 0)
- se n è negativo && $n >= -2^{N-1}$
  rappr(n) = $2^N - |n|$

esempio:

-796 quanti bit servono al minimo? 10?

no $-796 \notin [-2^9, 2^9 - 1] == [-512, 511]$

ne servono 11: $-796 \in [-2^{10}, 2^{10} - 1]$

quindi **rappr(-796)** = $2^{11} - 796 = 1252$

$(1252)_{10} = (\ ?\ )_2$

la rappresentazione di -796, con n=11 bit
è il numerale binario corrispondente al numero naturale 1252
(scritto su almeno 11 bit) ... suvvia, scrivi quel numerale ...

Rappresentazione in Complemento a 2 (esempio)

(N=numero di bit ... sufficiente a che n sia rappresentabile)

Dato un numero n,

- se n è positivo && n <= $2^{N-1}-1$ **rappr(n)** = normale numerale binario in N bit (il primo bit è necessariamente 0)
- se n è negativo && n >= 2^{N-1} **rappr(n)** = $2^N - |n|$

esempio: -796

quanti bit servono al minimo? 10? no $-796 \notin [-2^9, 2^9-1]$

ne servono 11: $-796 \in [-2^{10}, 2^{10}-1]$

rappr(-796) = $2^{11} - 796 = 1252$

$(1252)_{10} = (10011100100)_2$

1252/2 =	626	313	156	78	39	19	9	4	2	1	0
	0	0	1	0	0	1	1	1	0	0	1

provare con N=14 invece che 11...
che numerale viene?

Rappresentazione in Complemento a 2 (esempio)

(N=numero di bit ... sufficiente a che n sia rappresentabile)

Dato un numero n,

- se n è positivo && n <= $2^{N-1}-1$ **rappr(n)** = normale numerale binario in N bit (il primo bit è necessariamente 0)
- se n è negativo && n >= 2^{N-1} **rappr(n)** = $2^N - |n|$

esempio: -796

quanti bit servono al minimo? 10? no $-796 \notin [-2^9, 2^9 -1]$

ne servono 11: $-796 \in [-2^{10}, 2^{10} -1]$

rappr(-796) = $2^{11} - 796 = 1252$

$(1252)_{10} = (\ 10011100100\ )_2$

1252/2 = 626 313 156 78 39 19 9 4 2 1 0

0 0 1 0 0 1 1 1 0 0 1

su 32 bit

796	00000000000000000000001100011100
-796	11111111111111111111110011100100

provare con N=14 ...

Rappresentazione in Complemento a 2 – metodo alternativo

(N=numero di bit … sufficiente a che n sia rappresentabile)

Dato un numero n,

- se (n è positivo) && $(n <= 2^{N-1} - 1)$ **rappr(n)** = normale numerale binario in N bit (il primo necessariamente 0)
- se (n è negativo) && $(|n| <= 2^{N-1})$ la **rappr(n)** si ottiene
  - rappresentando in binario puro il numero positivo corrispondente (|n|)
  - invertendo i bit di tale rappresentazione
  - e sommando 1

esempio: -796 NB per la rappresentazione binaria pura bastano 10 bit, ma per la rappr. in compl. a 2 ne servono 11

- $(796)_{10}$ =		1	1	0	0	0	1	1	1	0	0
- in 11 bit	0	1	1	0	0	0	1	1	1	0	0
- inversione	1	0	0	1	1	1	0	0	0	1	1
- + 1	1	0	0	1	1	1	0	0	1	0	0

- **rappr(-796) =** 10011100100

provare con N=14 …

su 32 bit

796 00000000000000000000001100011100

-796 11111111111111111111110011100100

Rappresentazione in Complemento a 2
DECODIFICA da rappr(n) a n …

(N=numero di bit)

Data la rappr(n), qual è n?

Si procede seguendo il metodo di rappresentazione al contrario …

- se n è positivo (cioè rappr(n) inizia con 0, si decodifica normalmente, come in binario puro
- se n è negativo (rappr(n) inizia con 1) n si ottiene
  - sottraendo 1
  - invertendo i bit
  - decodificando il numero binario normalmente

esempio: rappr(n) = 10110001 (è negativo)

```
- -79               1 0 1 1 0 0 0 1 -
- sottr 1                         1
                    1 0 1 1 0 0 0 0
- inversione        0 1 0 0 1 1 1 1
                    e questo ↑ è 79
```

Tipo base char

Rappresenta i caratteri (di solito ASCII) in un byte.
Nel byte (la variabile di tipo char) c'è un codice virtualmente compreso nell'intervallo [0, 255]

```
char crt;
      crt = 'f'
      printf ("BLA … %c\n", crt);
```

(Costante carattere, tra apici)

A(65), B(66), …, Z(90), …, a(97), …, z(122)
0(48), …, 9(57)
!(33), #(35), {(123), }(125), ~(126)

La codifica ASCII ha codici da 0 a 255; la codifica usata nel byte char è un po' diversa ma equivalente

- i codici [-128, -1] corrispondono ai caratteri ASCII di codice [128, 255]
- i codici [0, 127] corrispondono ai caratteri ASCII di codice [0, 127]

Ma la cosa è trasparente …

Tipo base char: codici interi (-128??)

equivalenti

```
il carattere corrispondente al codice -128 (128) è Ç
il carattere corrispondente al codice -127 (129) è ü
il carattere corrispondente al codice -126 (130) è é
il carattere corrispondente al codice  -85 (171) è ½
il carattere corrispondente al codice  -84 (172) è ¼
il carattere corrispondente al codice   -1 (255) è
il carattere corrispondente al codice    0 (  0) è
il carattere corrispondente al codice   21 ( 21) è §
il carattere corrispondente al codice   35 ( 35) è #
il carattere corrispondente al codice   36 ( 36) è $
il carattere corrispondente al codice   37 ( 37) è %
il carattere corrispondente al codice   38 ( 38) è &
il carattere corrispondente al codice   39 ( 39) è '
il carattere corrispondente al codice   40 ( 40) è (
il carattere corrispondente al codice   41 ( 41) è )
il carattere corrispondente al codice   48 ( 48) è 0
il carattere corrispondente al codice   64 ( 64) è @
il carattere corrispondente al codice   65 ( 65) è A
il carattere corrispondente al codice   90 ( 90) è Z
il carattere corrispondente al codice   97 ( 97) è a
il carattere corrispondente al codice  121 (121) è y
```

Tipo base char: codici interi

```
char crt;
int cod;
  printf ("caro/a utente, dammi un carattere ...\n");
  scanf("%c", &crt);
  printf ("caro/a utente, mi hai dato il carattere %c avente
                                      codice %d\n", crt, crt);
...  ...      scanf("%d", &cod);
  printf ("caro/a utente, mi hai dato il codice %d che corrisponde
                                   al carattere %c\n", cod, cod);
```

```
caro/a utente, dammi un carattere ...
f
caro/a utente, mi hai dato il carattere f avente codice 102
caro/a utente, dammi un codice di carattere ...
-52
caro/a utente, mi hai dato il codice -52 che corrisponde al carattere ╠
FINE
_
```

```
caro/a utente, dammi un carattere ...
f
caro/a utente, mi hai dato il carattere f avente codice 102
caro/a utente, dammi un codice di carattere ...
204
caro/a utente, mi hai dato il codice 204 che corrisponde al carattere ╠
FINE
```

Tipo base char: aritmetica

```
int main () {
  char crt,crt2='d';
…                            scanf("%c", &crt);
printf ("caro/a utente, …due … sono %c e %c\n", crt+1, crt+2);

printf ("invece i due predecessori …%c e %c\n", crt-2, crt-1);

printf ("moltiplicando per 2 %c si ottiene %c\n", crt, crt*2);

printf ("dividendo per 2 %c si ottiene %c\n", crt, crt/2);
crt=149;
…
printf ("il … divisione di %c per d è %c\n", crt, crt%crt2);
```

```
caro/a utente, dammi un carattere ...
f
caro/a utente, i successivi due caratteri sono g e h
invece i due predecessori immediati sono d e e
moltiplicando per 2 f si ottiene ╠
dividando per 2 f si ottiene 3
il resto della divisione di 149 per 100 e' 49
il resto della divisione di -107 per 100 e' -7
il resto della divisione di ò per d e' ·
FINE
```

Come abbiamo fatto? --- Tipo base char: codici interi (-128??)

equivalenti

```
il carattere corrispondente al codice -128 (128) è Ç
il carattere corrispondente al codice -127 (129) è ü
il carattere corrispondente al codice -126 (130) è é
il carattere corrispondente al codice  -85 (171) è ½
il carattere corrispondente al codice  -84 (172) è ¼
il carattere corrispondente al codice   -1 (255) è
il carattere corrispondente al codice    0 (  0) è
il carattere corrispondente al codice   21 ( 21) è §
il carattere corrispondente al codice   35 ( 35) è #
il carattere corrispondente al codice   36 ( 36) è $
il carattere corrispondente al codice   37 ( 37) è %
il carattere corrispondente al codice   38 ( 38) è &
il carattere corrispondente al codice   39 ( 39) è '
il carattere corrispondente al codice   40 ( 40) è (
il carattere corrispondente al codice   41 ( 41) è )
il carattere corrispondente al codice   48 ( 48) è 0
il carattere corrispondente al codice   64 ( 64) è @
il carattere corrispondente al codice   65 ( 65) è A
il carattere corrispondente al codice   90 ( 90) è Z
il carattere corrispondente al codice   97 ( 97) è a
il carattere corrispondente al codice  121 (121) è y
```

provare a scrivere un programma che realizza questo output ...

Tipo base char: codici interi

Con istruzione iterativa WHILE

```
#include <stdio.h>
int main () {
  char c;
  int cod, c1;

  cod = -128;               /* inizializzazione */
  while (cod <= 127) {
      if (cod<0)
         c1=cod+256;
      else c1 = cod;

      printf ("il car… %4d (%3d) …%c\n", cod, c1, cod);

      cod = cod + 1;               /* modifica variabile di test */
  }

printf("FINE\n");
return 0;
}
```

Visto che ci siamo … Espressione condizionale

C'è un altro operatore … OPERATORE CONDIZIONALE **? :**
è TERNARIO, cioè prende tre operandi, permettendo di ottenere un'espressione condizionale:

EXP_C (condizione) ? (exp1) : (exp2)

Valutazione di EXP_C :

se la condizione vale TRUE, EXP_C ritorna il valore di exp1;
altrimenti EXP_C ritorna il valore di exp2.

```
cod = -128;                /* inizializzazione */
while (cod <= 127) {
    printf ("il carattere corrispondente al
            codice %4d (%3d) è %c\n",
            cod, (cod<0? cod+256 : cod), cod);

    cod = cod + 1;                /* modifica variabile di test */
}
```

Visto che ci siamo … Obfuscation

Se usiamo contemporaneamente l'espressione condizionale e quella di post incremento otteniamo in questo caso un codice molto più compresso (corto) e forse più difficile da comprendere

EXP_C (condizione) ? (exp1) : (exp2)
se la condizione vale TRUE, EXP_C ritorna il valore di exp1;
altrimenti EXP_C ritorna il valore di exp2.

post-incremento cod++
(restituisce il valore di cod prima di incrementarlo)

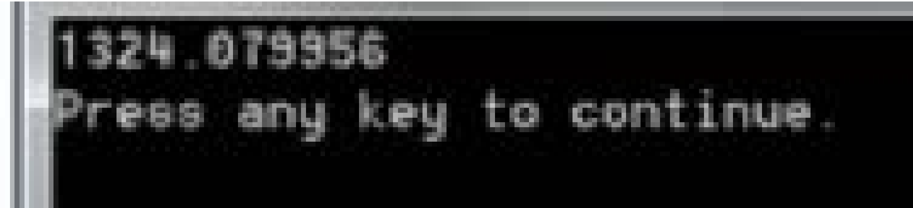
```
cod = -128;                /* inizializzazione */
while (cod <= 127)
    printf ("il carattere corrispondente al
            codice %4d (%3d) è %c\n",
            cod, (cod<0? cod+256 : cod), cod++);
```

```
/* la modifica della variabile di test avviene nella valutazione del
postincremento */
```

Tipo base float

Rappresentazione approssimata dei numeri reali

```
float r;
r=1324.08;
printf("%f\n", r);
```



Si chiama RAPPRESENTAZIONE IN VIRGOLA MOBILE (Floating Point)

La rappresentazione deriva dalla notazione scientifica, in cui un numero è rappresentato come

$$M \times 10^{E}$$

M è la ***mantissa***, che rappresenta la parte frazionaria del numero, con (quasi) tutte le sue cifre;
E è ***l'esponente***.

$$123.365 \times 10^6 = 123365000 = 1.23365 \times 10^8 = 0.123365 \times 10^9$$

NB1
Ovviamente nella rappresentazione FP non tratteremo più potenze di 10, ma solo potenze di 2 ...

NB2
La rappresentazione FP è «normalizzata»: prima del «point» c'è una cifra non nulla (cioè 1)

Tipo base float

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per le cifre della mantissa normalizzata, escluso il primo 1 (M)

Nella locazione di memoria ci sono le seguenti informazioni:

- **segno** s del numero rappresentato (0/positivo, 1/negativo)

- **M** rappresenta la sequenza di cifre (binarie) della **mantissa normalizzata** … solo quelle successive al "1." … vedi gli esempi dopo …

- **esponente** E: un numero (binario) compreso tra 0 e 255; scritto in "eccesso 127"
  … [0,255] -> [-128, 127]

Tipo base float

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per le cifre della mantissa normalizzata, escluso il primo 1 (M)

Nella locazione di memoria ci sono le seguenti informazioni:
- segno s del numero rappresentato (0/positivo, 1/negativo)

- M rappresenta la sequenza di cifre (binarie) della mantissa normalizzata … solo quelle successive al "1." … vedi gli esempi dopo …

- esponente E: un numero (binario) compreso tra 0 e 255; questo numero è scritto in "eccesso 127", cioè è stato calcolato aggiungendo 127 al suo valore effettivo.
  Quindi, quando viene usato nel calcolo $M \times 2^{esp}$, gli viene sottratto 127.
  Questo permette di avere esponenti anche negativi, tra -127 a 128, e quindi di rappresentare numeri reali di ordini di grandezza che vanno da 2^{-127} a 2^{128} … vedi gli esempi dopo …

Tipo base float

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per M

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Nella locazione di memoria ci sono le seguenti informazioni:
- segno s del numero rappresentato (0/positivo, 1/negativo)

- esponente E: un numero (binario) compreso tra 0 e 255; questo numero è scritto in "eccesso 127", cioè è stato calcolato aggiungendo 127 al suo valore effettivo. Quindi, quando viene usato nel calcolo indicato sopra, gli viene sottratto 127. Questo permette di avere esponenti anche negativi, tra -127 a 128, e quindi di rappresentare numeri reali di ordini di grandezza che vanno da 2^{-127} a 2^{128} … vedi gli esempi dopo …

- M rappresenta la sequenza di cifre (binarie) della mantissa normalizzata … solo quelle successive al "1." … vedi gli esempi dopo …

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per M

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

$(4.25)_{10}$ = 100.01

→ mantissa normalizzata 1.0001

M = 0001 (solo le cifre della mantissa normalizzata successive al "1.")

→ esponente di 2 per tornare dalla mantissa normalizzata al numero: 2

(a questo esponente va aggiunto 127, per avere la sua rappresentazione in eccesso 127)

$E = 2+127 = 129\ (10000001)_2$

rappr(4.25) = 0 10000001 00010000000000000000000

prova: facciamo il calcolo indicato in alto a destra, per provare che la rappresentazione FP qui sopra è corretta ...

$$\mathbf{1.M \times 2^{E-127}} = 1.0001 \times 2^2 = \mathbf{4.25}$$

☺ sviluppa il calcolo

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Altro esempio
data la rappresentazione
1 10000000 10010010000000000000000
qual è il numero?

segno ?

E = ?
M = ?
mantissa normalizzata = ?

numero =

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Altro esempio

data la rappresentazione

1 10000000 10010010000000000000000

qual è il numero?

(usiamo il metodo qui in alto a destra ...)

segno negativo;

E = 128 = 1 + 127 cioè E è 128, ma il valore effettivo è 128 in eccesso 127 cioè 1;

M = 1001001

mantissa normalizzata = 1.1001001 = 1 + ½ + 1/16 + 1/128 = 1.5703125

$$\text{numero} = -1.5703125 \times 2^1 = -3.140625$$

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = …

mantissa normalizzata = …

M =

E =

rappr(5.375) = 0 . . . .

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = ...

M =

E =

rappr(5.375) = 0 . . . .

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = 1.01011

M =

E =

rappr(5.375) = 0 . . . .

Tipo base float - esempi

Su 32 bit si usano

- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = 1.01011

M = 01011 (la parte che va nei 23 bit ...)

E =

rappr(5.375) = 0 . . . . ☺

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = 1.01011

M = 01011 (la parte che va nei 23 bit ...)

E = 2 + 127 = $(☺)_{10}$ = $(☺)_2$

rappr(5.375) = 0 . . . . ☺

Tipo base float - esempi

Su 32 bit si usano

- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = 1.01011

M = 01011 (la parte che va nei 23 bit ...)

E = 2 + 127 = $(129)_{10}$ = $(☺)_2$

rappr(5.375) = 0 . . . . ☺

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = 1.01011

M = 01011 (la parte che va nei 23 bit ...)

$E = 2 + 127 = (129)_{10} = (10000001)_2$

rappr(5.375) = 0 . . . .

Tipo base float - esempi

Su 32 bit si usano

- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = 1.01011

M = 01011 (la parte che va nei 23 bit ...)

$E = 2 + 127 = (129)_{10} = (10000001)_2$

rappr(5.375) = 0 10000001 01011000000000000000000

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

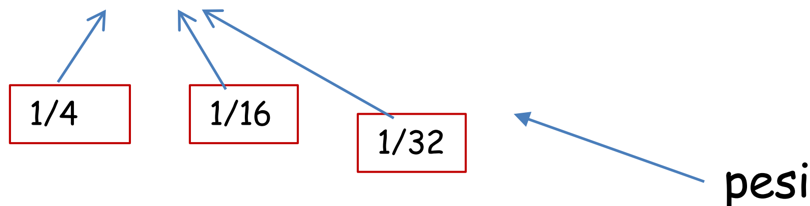
Dato $(5.375)_{10}$

mantissa normalizzata = 1.01011

M = 01011 (la parte che va nei 23 bit …)

E = 2 + 127 = $(129)_{10}$ = $(10000001)_2$

rappr(5.375) = 0 10000001 01011000000000000000000



prova: facciamo il calcolo indicato in alto a destra, per provare che la rappresentazione FP qui sopra è corretta …

$$1.M\times2^{E-127} = \ldots\ldots\ldots\ldots\ldots = 5.375$$

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(796.25)_{10}$

Rappresentazione in binario = ...

mantissa normalizzata = ...

M =

E =

rappr(796.25) = 0 . . . .

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(796.25)_{10}$

Rappresentazione in binario = 1100011100.01

mantissa normalizzata = 1.10001110001

M = 10001110001

E = 9+127 = 136 $(10001000)_2$

rappr(796.25) = 0 10001000 10001110001000000000000

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(796.25)_{10}$

Rappresentazione in binario = 1100011100.01

mantissa normalizzata = 1.10001110001

M = 10001110001

E = 9+127 = 136 $(10001000)_2$

rappr(796.25) = 0 10001000 10001110001000000000000

1/2 1/32 1/64 1/128 1/2048

prova: facciamo il calcolo indicato in alto a destra, per provare che la rappresentazione FP qui sopra è corretta ...

$1.M \times 2^{E-127}$ = ... = **796.25**

Tipo base float - sparsity

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

su 64 bit
- E è di 11 bit e M di 52;
- è eccesso 1023
- ordini di grandezza da 2^{-1023} a 2^{1024}

Gli ordini di grandezza dei numeri rappresentati sono ingenti (da 2^{-127} a 2^{128})

ma i numeri rappresentati tendono ad essere sempre più sparsi, man mano che crescono di grandezza:

infatti, per ogni esponente, c'è un numero costante di numerali (e quindi di numeri rappresentati): sono 2^{23}, se la mantissa è di 23 bit

… più i numeri sono grandi, più sono sparsi …

Esercizi – numeri binari

Quali numeri naturali sono rappresentati dai seguenti numerali binari?

100010 e qual è il più grande numero rappresentabile con questo numero di cifre?

1101 e qual è il più grande numero rappresentabile con questo numero di cifre?

111101 e qual è il più grande numero rappresentabile con questo numero di cifre?

10101001 e qual è il più grande numero rappresentabile con questo numero di cifre?

10010110000000 e qual è il più grande numero rappresentabile con questo numero di cifre?

101000000101000 e qual è il più grande numero rappresentabile con questo numero di cifre?

101001101 e qual è il più grande numero rappresentabile con questo numero di cifre?

Per questi numeri, poi, scrivere la rappresentazione ottale e quella esadecimale.

Per i seguenti numeri decimali, scrivere la rappresentazione in binario puro.

0.703125 125.1875 132.6875 912.75 1216.1875 5509.8125

Per i seguenti numeri naturali, rappresentati in forma decimale, stabilire il minimo numero di cifre necessarie per la rappresentazione in binario puro, e poi scrivere la rappresentazione medesima.

151 212 918 96000 31718 55 181 987987 128128128

Esercizi – complemento a due

Per i seguenti numeri interi, stabilire il minimo numero di cifre necessarie per la rappresentazione in complemento a due, e poi scrivere la rappresentazione medesima.

19   -19   65   715   -716   32000   -231231

Riscrivere poi tutti i numeri, in complemento a due su 32 bit.

(Quanti '1' contiene -231231?

scrivilo, prima di guardare la prossima slide

Esercizi – complemento a due

Per i seguenti numeri interi, stabilire il minimo numero di cifre necessarie per la rappresentazione in complemento a due, e poi scrivere la rappresentazione medesima.

19 -19 65 715 -716 32000 -231231

Riscrivere poi tutti i numeri, in complemento a due su 32 bit.

(Quanti '1' contiene -231231?

meno di 15? Sbagliato.

Si intende quanti 1 contiene la rappresentazione in complemento a due di 231231)

Esercizi – Floating Point

Calcolare il numero razionale rappresentato dalle seguenti notazioni FP (su 32 bit, 23 di mantissa)

11000100010001110001000000000000 (noto …)
11000100011001110001000000000000 (piccole grandi differenze)
11000000101011000000000000000000 (noto …)
01000100101011000010010000000000
11000010010011101100000000000000

Per i seguenti numeri razionali, scrivere la rappresentazione in FP (32 bit).

128796.70312500
125.1875
796.25 (noto …)
5509.8125

Esercizi – Floating Point

Calcolare il numero razionale rappresentato dalle seguenti notazioni FP (su 32 bit, 23 di mantissa)

```
11000100010001110001000000000000 -796,25
11000100011001110001000000000000 -924,25
11000000101011000000000000000000 -5,375
01000100101011000010010000000000 1377,125
11000010010011101100000000000000 -51,6875
```

Per i seguenti numeri razionali, scrivere la rappresentazione in FP (32 bit).

```
128796.70312500     01000111111110111000111001011010
125.1875            01000010111110100110000000000000
796.25              01000100010001110001000000000000
5509.8125           01000101101011000010111010000000
```