

FONDAMENTI DI INFORMATICA

Alessandro Renda

Dipartimento di Ingegneria e Architettura, Università degli Studi di Trieste

RAPPRESENTAZIONE DEI NUMERI

Anno Accademico 2024/2025

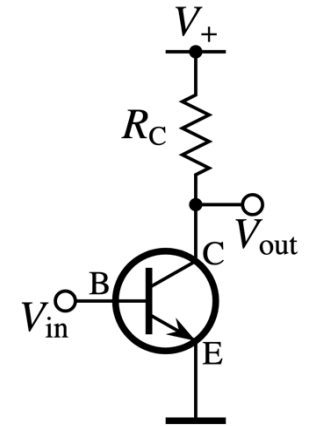
Introduzione

Codifica binaria

- Necessità di ottenere una **rappresentazione dell'informazione** su cui il sistema di calcolo sia in grado di operare
- La codifica usata nei calcolatori è quella binaria
- Convenzionalmente, i simboli usati sono 0 e 1
- La cifra binaria prende il nome di *bit* (*Binary digit*)

Codifica binaria: perchè

- *Transistor*: componente elettronico che può essere usato come interruttore
 - Quando alla base B si applica un'opportuna tensione V_i , l'interruttore è chiuso
 - La corrente fluisce fra collettore C ed emettitore E
 - La tensione in uscita V_{out} è nulla
 - Quando alla base B la tensione applicata è nulla, l'interruttore è aperto
 - La corrente non fluisce fra collettore C ed emettitore E
 - La tensione in uscita V_{out} è pari a V_+
- $\sim 10\,000\,000\,000$ (decine / centinaia di miliardi) transistor in un nostro PC



Codifica binaria: perchè

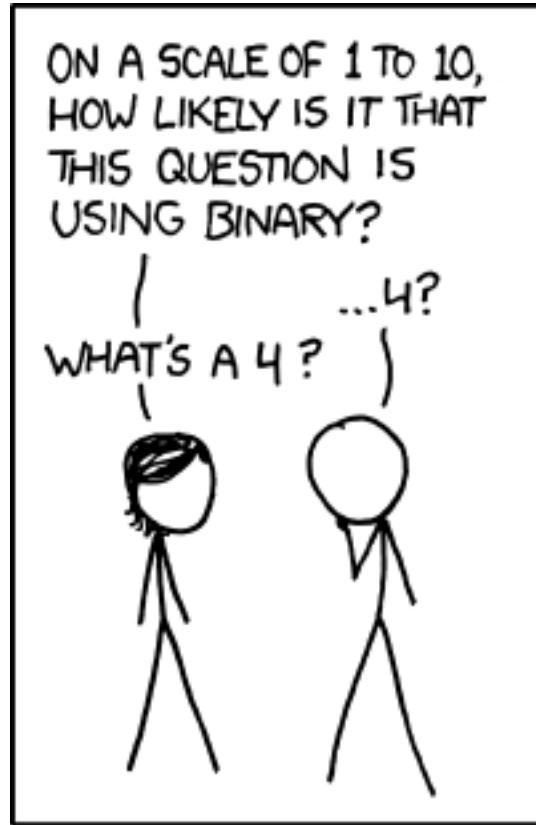
- Si potrebbero rappresentare elettronicamente più cifre? Sì, ma...
 - Maggiore consumo energetico
 - Maggiore dissipazione del calore
 - Minore affidabilità dei dispositivi
 - Minore velocità di funzionamento
- Il transistor permette di *rappresentare* 0 e 1 elettronicamente
 - Costo relativamente basso, efficienza relativamente alta

Codifica binaria

- Codifiche notevoli

- | | | |
|---------------------|--|----------------|
| • Binaria | $0, 1$ | 1101_2 |
| • Decimale | $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ | 13_{10} |
| • Esadecimale (HEX) | $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$ | $D_{16} = 0xD$ |

- In pratica, i simboli dell'alfabeto che vengono usati in una generica base b sono i primi b della sequenza $(0, 1, \dots, 9, A, B, \dots, Z)$
 - dove $A = 10$, $B = 11$ e così via



<https://xkcd.com/953/>

Rappresentazione dei numeri naturali

Rappresentazione dei numeri naturali

1990_{10}

- Rappresentazione in *base 10* di un valore
- «1» è detta *cifra più significativa*
- «0» è detta *cifra meno significativa*

Rappresentazione dei numeri naturali

- Il codice sfrutta un **alfabeto**: insieme finito di b elementi, da 0 a $b - 1$
- **Notazione posizionale**: il codice per rappresentare i numeri è basato sulla posizione delle cifre

$$1990_{10} = 1 \cdot 1000 + 9 \cdot 100 + 9 \cdot 10 + 0$$

$$1990_{10} = 1 \cdot 10^3 + 9 \cdot 10^2 + 9 \cdot 10^1 + 0 \cdot 10^0$$

- Ogni cifra è moltiplicata per la potenza della costante b detta **base**
- Nella determinazione del valore, ogni cifra della rappresentazione assume un peso che dipende dalla posizione in cui è collocata

Rappresentazione dei numeri naturali

- Con k cifre e un alfabeto con b simboli si possono rappresentare b^k elementi
- L'intervallo di valori che possiamo rappresentare è $[0 \dots b^k - 1]$

Esempio: $b = 10, k = 3$

000	001	002	003	...				998	999
-----	-----	-----	-----	-----	--	--	--	-----	-----

Conversioni di base: da base b a 10

- «Ogni cifra è moltiplicata per la potenza di una costante b detta **base**»

$$x = \sum_{i=0}^{k-1} a_i \cdot b^i = a_{k-1} \cdot b^{k-1} + \dots + a_1 \cdot b^1 + a_0 \cdot b^0$$

Esempio:

$$\begin{aligned} 21011_3 &= 2 \cdot 3^4 + 1 \cdot 3^3 + 0 \cdot 3^2 + 1 \cdot 3^1 + 1 \cdot 3^0 \\ &= 2 \cdot 81 + 1 \cdot 27 + 0 \cdot 9 + 1 \cdot 3 + 1 \cdot 1 \\ &= 193_{10} \end{aligned}$$

Conversioni di base: da base b a 10

Esempio:

$$\begin{aligned} 1B_{12} &= 1 \cdot 12^1 + 11 \cdot 12^0 \\ &= 1 \cdot 12 + 11 \cdot 1 \\ &= 23_{10} \end{aligned}$$

Esercizio

Esempio:

$$3F4_{16} =$$

Esercizio

Esempio:

$$\begin{aligned} 3F4_{16} &= 3 \cdot 16^2 + 15 \cdot 16^1 + 4 \cdot 16^0 \\ &= 3 \cdot 256 + 15 \cdot 16 + 4 \cdot 1 \\ &= 1012_{10} \end{aligned}$$


Conversioni di base: da base 10 a base b

- Si calcola la divisione intera tra il numero (e i successivi quozienti) e la base
- Si registrano i resti delle divisioni fino a quando il quoziente è nullo
- Il valore in base b è costituito dai **resti letti al contrario**

Conversioni di base: da base 10 a base b

Esempio: trasformazione del valore 22_{10} in base $b = 2$

valore	base	quoziente	resto
22	2	11	0
11	2	5	1
5	2	2	1
2	2	1	0
1	2	0	1




$$22_{10} = 10110_2$$

Conversioni di base: da base 10 a base b

Esempio: trasformazione del valore 179_{10} in base $b = 12$

valore	base	quoziente	resto
179	12	14	11
14	12	1	2
1	12	0	1



$$179_{10} = 12B_{12}$$

Conversioni di base: da base 10 a base b

$$\begin{aligned}
 x &= \sum_{i=0}^{k-1} a_i \cdot b^i = a_{k-1} \cdot b^{k-1} + \dots + a_1 \cdot b^1 + a_0 \cdot b^0 \\
 &= a_0 + \sum_{i=1}^{k-1} a_i \cdot b^i \\
 &= a_0 + b \cdot \sum_{i=1}^{k-1} a_i \cdot b^{i-1} \\
 &= a_0 + b \cdot q_1
 \end{aligned}$$

$$\text{con } 0 \leq a_0 < b \text{ e } q_1 = \sum_{i=1}^{k-1} a_i \cdot b^{i-1} > 0$$

Conversioni di base: da base 10 a base b

$$x = a_0 + b \cdot q_1$$

- Il valore a_0 è il **resto** della divisione intera x/b

Conversioni di base: da base 10 a base b

Dato $x = a_0 + b \cdot q_1$, analogamente si ricava:

$$q_1 = \sum_{i=1}^{k-1} a_i \cdot b^{i-1} = a_1 + b \cdot \sum_{i=2}^{k-1} a_i \cdot b^{i-1} = a_1 + b \cdot q_2$$

$$q_2 = \sum_{i=2}^{k-1} a_i \cdot b^{i-1} = a_2 + b \cdot \sum_{i=3}^{k-1} a_i \cdot b^{i-1} = a_2 + b \cdot q_3$$

...

- I valori a_i (con $i > 0$) sono i **resti** della divisione intera q_i/b , dove il valore q_i è il **quoziente** di q_{i-1}/b

Conversioni di base: da base 10 a base b

- L'ultimo coefficiente (ovvero la cifra più significativa) si ottiene quando si arriva ad avere un quoziente

$$a_{k-1} = q_{k-1} < b$$

- Il valore in base b è costituito dai **resti letti al contrario** $a_{k-1}, a_{k-2}, \dots, a_1, a_0$

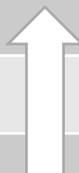
Esercizio

Esempio: trasformazione del valore 672_{10} in base $b = 16$

Esercizio

Esempio: trasformazione del valore 672_{10} in base $b = 16$

valore	base	quoziente	resto
672	16	42	0
42	16	2	10
2	16	0	2



$$672_{10} = 2A0_{16}$$

Conversioni di base: da base 2 a base 16

- In certi ambiti il numero binario può essere lungo e scomodo da usare. Esempi:
 - MAC address: codice di 48 bit associato ad ogni dispositivo di rete
 - Codifica RGB: codice di 24 bit per la rappresentazione di un colore a partire dai primari
- Si raggruppano le cifre a gruppi di 4, e si convertono in esadecimale ($2^4 = 16$)

Esempio:

$$\begin{aligned} 10011011_2 &= 1001 \ 1011 \\ &= 9B_{16} \end{aligned}$$

Conversioni di base: da base 16 a base 2

- A ogni cifra esadecimale si fa corrispondere una sequenza di quattro bit
- Infine si concatenano le sequenze di bit ottenute

Esempio:: trasformazione del valore $FA5_{16}$ in base $b = 2$

$$F_{16} = 15_{10} = 1111_2$$

$$A_{16} = 10_{10} = 1010_2$$

$$5_{16} = 5_{10} = 0101_2$$

$$FA5_{16} = 1111\ 1010\ 0101_2$$

Python

- Da decimale a binario, ottale, esadecimale

```
>>> bin(42)
'0b101010'
```

```
>>> oct(42)
'0o52'
```

```
>>> hex(42)
'0x2a'
```

- Da binario, ottale, esadecimale a decimale

```
>>> int("101010", 2)
42
```

```
>>> int("52", 8)
42
```

```
>>> int("2a", 16)
42
```

Operazioni aritmetiche con notazione posizionale

- Somma:
 - si procede da destra a sinistra, riportando l'eccedenza sulla colonna successiva

Esempio:

$$137_{10} + 492_{10} = 629_{10}$$

eccedenza	1			
x	1	3	7	+
y	4	9	2	=
$x + y$	6	2	9	

Esempio:

$$0101_2 + 1001_2 = 1110_2$$

eccedenza			1		
x	0	1	0	1	+
y	1	0	0	1	=
$x + y$	1	1	1	0	

Operazioni aritmetiche con notazione posizionale

- Somma:
 - Si procede da destra a sinistra, riportando l'eccedenza sulla colonna successiva
 - Se le cifre non sono sufficienti per rappresentare il valore del risultato: *overflow*

Esempio: supponiamo di avere 4 bit a disposizione

$$1101_2 + 1001_2 = 0110_2$$

$$13 + 9 = 6$$

Overflow

- **Overflow aritmetico**

- Errore che si verifica quando il **risultato di un'operazione** non è rappresentabile con la medesima codifica e il numero di cifre degli operandi
- Si verifica nella somma di numeri positivi:

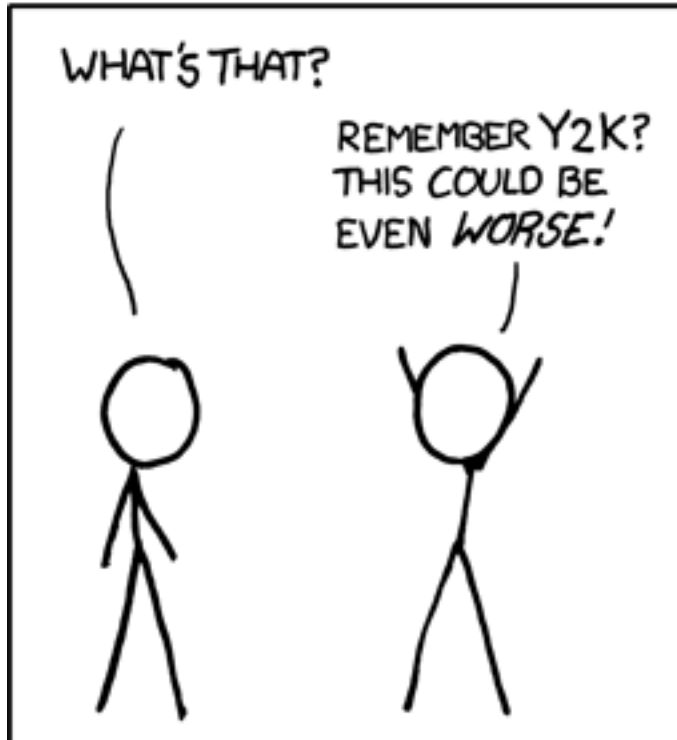
- Quando ho un numero fisso di bit a disposizione
- Quando ho un riporto sul bit più significativo

Possiamo *accorgercene*

Python sfrutta questa possibilità e garantisce che l'overflow fra interi non si verifichi mai

Limiti nella rappresentazione dei dati

I'M GLAD WE'RE SWITCHING TO 64-BIT, BECAUSE I WASN'T LOOKING FORWARD TO CONVINCING PEOPLE TO CARE ABOUT THE UNIX 2038 PROBLEM.



<https://xkcd.com/607/>



Y2K problem
aka "Millennium Bug"



https://en.wikipedia.org/wiki/Year_2000_problem#Documented_errors

Binary : 01111111 11111111 11111111 11110000

Decimal : 2147483632

Date : 2038-01-19 03:13:52 (UTC)

Date : 2038-01-19 03:13:52 (UTC)

Y2K38 problem

Codifica binaria

- Le prime potenze di 2

2^0	1		
2^1	2	2^{11}	2048
2^2	4	2^{12}	4096
2^3	8	2^{13}	8192
2^4	16	2^{14}	16384
2^5	32	2^{15}	32768
2^6	64	2^{16}	65536
2^7	128	2^{17}	131072
2^8	256	2^{18}	262144
2^9	512	2^{19}	524288
2^{10}	1024	2^{20}	1048576

Codifica binaria

- Grandezze fondamentali

simbolo	nome	valore	
b	bit	1 bit: 0 o 1	
B	byte	8 bit	
kB	kilobyte	2^{10} byte	2^{10} byte
MB	megabyte	2^{10} kilobyte	2^{20} byte
GB	gigabyte	2^{10} megabyte	2^{30} byte
TB	terabyte	2^{10} gigabyte	2^{40} byte
PB	petabyte	2^{10} terabyte	2^{50} byte
EB	exabyte	2^{10} petabyte	2^{60} byte
ZB	zettabyte	2^{10} exabyte	2^{70} byte
YB	yottabyte	2^{10} zettabyte	2^{80} byte

SI	Differenza %
10^3	2.40
10^6	4.86
10^9	7.37
10^{12}	9.95
10^{15}	12.59
10^{18}	15.29
10^{21}	18.06
10^{24}	20.89

Rappresentazione dei numeri interi

Codifica binaria dei numeri interi

- Di seguito, si presentano pregi e difetti di due codifiche
 - Modulo e segno
 - Complemento a due: tipicamente adottata nei calcolatori elettronici

Codifica binaria dei numeri interi: modulo e segno

- Si codifica il segno sul bit più significativo. I restanti bit codificano il modulo.

$b = 2$	$b = 10$	$b = 2$	$b = 10$
000	0	100	-0
001	1	101	-1
010	2	110	-2
011	3	111	-3

- *Osservazioni:*
 - Con k cifre, consente di rappresentare i numeri da $-(2^{k-1} - 1)$ a $+(2^{k-1} - 1)$
 - La codifica per valori positivi coincide con la codifica binaria naturale
 - La rappresentazione dello 0 è duplice (nell'esempio con $k = 3$: {000, 100})



Codifica binaria dei numeri interi: modulo e segno

- *Osservazioni (continua):*
 - Non consente di svolgere velocemente le principali operazioni aritmetiche: non si possono usare le stesse regole usate per i numeri naturali

x_{10}	x_{c2}
-3	111
-2	110
-1	101
-0	100
0	000
1	001
2	010
3	011

Esempio: *come non si può fare*

eccedenza				
$x = -2$	1	1	0	+
$y = +1$	0	0	1	=
$x + y = -1$	1	1	1	

Codifica binaria dei numeri interi: complemento a 2

- Il bit più significativo rappresenta il segno, è fattore additivo e ha peso -2^{k-1}

x_{10}	x_{c2}
0	000
1	001
2	010
3	011

x_{10}	x_{c2}
-1	111
-2	110
-3	101
-4	100

Esempio:

$$010_{c2} = -0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\ = 2_{10}$$

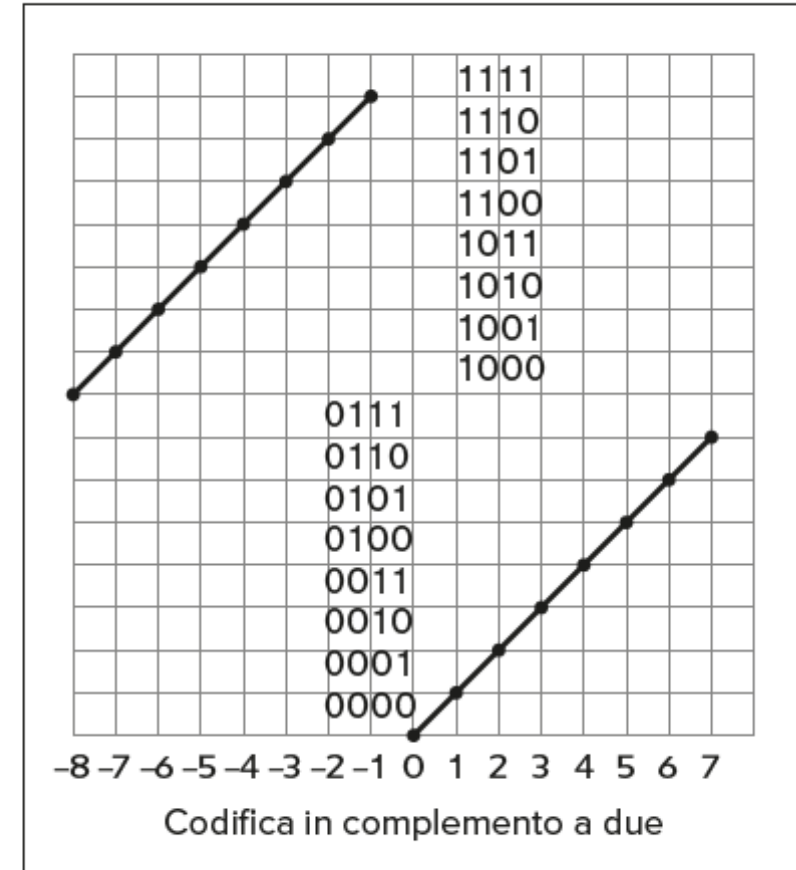
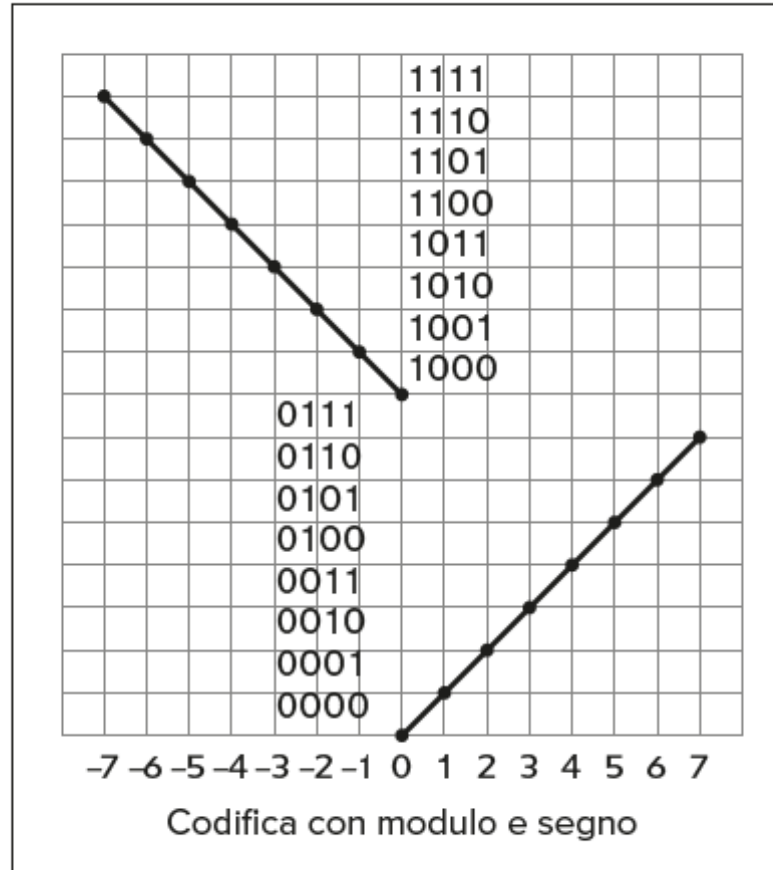
Esempio:

$$110_{c2} = -1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\ = -2_{10}$$

Codifica binaria dei numeri interi

Figura 2.8

Interpretazione grafica delle rappresentazioni dei numeri interi.



Esercizio

Esempio: trasformazione del valore 10101101 in base $b = 10$

Esempio: trasformazione del valore 00101101_2 in base $b = 10$

Esempio: trasformazione del valore 00101101_{C2} in base $b = 10$

Esempio: trasformazione del valore 10101101_{C2} in base $b = 10$

Esempio: trasformazione del valore 01111111_{C2} in base $b = 10$

Esempio: trasformazione del valore 10000000_{C2} in base $b = 10$

Esempio: trasformazione del valore 11111111_{C2} in base $b = 10$

Esempio: trasformazione del valore 00000000_{C2} in base $b = 10$

Codifica binaria dei numeri interi: complemento a 2

- Altra interpretazione: dato x il numero da rappresentare e k il numero di bit a disposizione, si codifica x con il valore binario naturale corrispondente a $2^k + x$

x_{10}	$(2^3 + x)_{10}$	$(2^3 + x)_2$	x_{c2}
0	8	1000	000
1	9	1001	001
2	10	1010	010
3	11	1011	011

x_{10}	$(2^3 + x)_{10}$	$(2^3 + x)_2$	x_{c2}
-1	7	0111	111
-2	6	0110	110
-3	5	0101	101
-4	4	0100	100

Codifica binaria dei numeri interi: complemento a 2

- *Osservazioni:*
 - Consente di rappresentare i numeri da $-(2^{k-1})$ a $+(2^{k-1} - 1)$
 - La codifica per x positivi coincide con la codifica binaria naturale
 - C'è un'unica rappresentazione dello 0

Codifica binaria dei numeri interi: complemento a 2

- *Osservazioni (continua):*
 - La codifica in complemento a 2 per x negativi può essere svolta **meccanicamente**
 - Si rappresenta il numero negativo binario senza tenere conto del segno
 - Si invertono i valori dei singoli bit
 - Si aggiunge 1 al risultato

x_{10}	$ x _2$	Inverto	Aggiungo 1
-1	001	110	111
-2	010	101	110
-3	011	100	101
-4	100	011	100

Codifica binaria dei numeri interi: complemento a 2

- *Osservazioni (continua):*

- Il bit più significativo rappresenta il segno: è un fattore additivo e ha peso -2^{k-1}
- Consente di svolgere velocemente le principali operazioni aritmetiche:

x_{10}	x_{c2}
-4	100
-3	101
-2	110
-1	111
0	000
1	001
2	010
3	011

Esempio:

eccedenza				
$x = -2$	1	1	0	+
$y = +1$	0	0	1	=
$x + y$	1	1	1	

Codifica binaria dei numeri interi: complemento a 2

- Osservazioni (continua):

- Il bit più significativo rappresenta il segno: è un fattore additivo e ha peso -2^{k-1}
- Consente di svolgere velocemente le principali operazioni aritmetiche:

Esempio:

x_{10}	$ x _2$	Invertito	Aggiungo 1
-25	0001 1001	1110 0110	1110 0111
-65	0100 0001	1011 1110	1011 1111
-90	0101 1010	1010 0101	1010 0110

eccedenza	1	1	1	1	1	1	1		
-25	1	1	1	0	0	1	1	1	+
-65	1	0	1	1	1	1	1	1	=
$x + y$	1	0	1	0	0	1	1	0	

Nota: il bit in eccesso viene scartato

Operazioni aritmetiche in complemento a 2

Esempio:

x_{10}	x_{c2}
-25	1110 0111
+65	0100 0001
+40	0010 1000

eccedenza	1				1	1	1		
-25	1	1	1	0	0	1	1	1	+
+65	0	1	0	0	0	0	0	1	=
$x + y$	0	0	1	0	1	0	0	0	

Nota: il bit in eccesso viene scartato

Overflow in complemento a 2

- Il riporto sul bit più significativo non è indicativo della condizione di *overflow*
- L'overflow è associato ad un'**anomalia di segno**
- Consideriamo il caso della somma di due numeri A e B espressi in C2
 - Se A e B hanno segno diverso (**discorde**): non può esserci overflow
 - Se A e B hanno lo stesso segno (**concorde**)
 - $A \geq 0, B \geq 0, A + B \geq 0$
 - $A \geq 0, B \geq 0, A + B < 0$
 - $A < 0, B < 0, A + B < 0$
 - $A < 0, B < 0, A + B \geq 0$



Overflow

The diagram consists of a central rectangular box labeled 'Overflow'. To its left, there are four bullet points. The second and fourth bullet points are enclosed in orange rectangular boxes. Two orange lines originate from the right side of the second box and point towards the 'Overflow' box. Another two orange lines originate from the right side of the fourth box and also point towards the 'Overflow' box, indicating that these two specific cases lead to an overflow.

Overflow in complemento a 2

Esempio:

eccedenza		1	1						
80	0	1	0	1	0	0	0	0	+
52	0	0	1	1	0	1	0	0	=
$x + y$	1	0	0	0	0	1	0	0	

Anomalia di segno indica overflow

Overflow in complemento a 2

Esempio:

eccedenza									
-80	1	0	1	1	0	0	0	0	+
-52	1	1	0	0	1	1	0	0	=
$x + y$	0	1	1	1	1	1	0	0	

Anomalia di segno indica overflow

Overflow in complemento a 2

Esempio:

eccedenza		1	1						
-80	1	0	1	1	0	0	0	0	+
+52	0	0	1	1	0	1	0	0	=
$x + y$	1	1	1	0	0	1	0	0	

Gli operandi hanno segno discorde: non può esserci overflow

Operazioni elementari sui bit

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

	<i>NOT</i>
0	1
1	0

Negazione

<i>AND</i>	0	1
0	0	0
1	0	1

Congiunzione logica

<i>OR</i>	0	1
0	0	1
1	1	1

Disgiunzione inclusiva

<i>XOR</i>	0	1
0	0	1
1	1	0

Disgiunzione esclusiva

Rappresentazione dei numeri reali

Rappresentazione di numeri reali

- Di seguito, si presentano pregi e difetti di due codifiche
 - Virgola fissa
 - Virgola mobile

Rappresentazione di numeri reali: virgola fissa

- Il sistema di calcolo utilizza k cifre per la rappresentazione
 - Le f cifre più a destra sono dedicate alla parte frazionaria
 - Le $k - f$ cifre più a sinistra sono dedicate alla parte intera
- Per la rappresentazione della *parte intera* x_I : metodo della divisione (già visto)
- Per la rappresentazione della *parte frazionaria* x_F :

$$x_F = a_{-1} \cdot b^{-1} + a_{-2} \cdot b^{-2} + a_{-3} \cdot b^{-3} + \dots$$

$$\begin{aligned} b \cdot x_F &= b \cdot (a_{-1} \cdot b^{-1} + a_{-2} \cdot b^{-2} + a_{-3} \cdot b^{-3} + \dots) \\ &= a_{-1} + a_{-2} \cdot b^{-1} + a_{-3} \cdot b^{-2} + \dots \end{aligned}$$

- a_{-1} si ottiene come parte intera di $b \cdot x_F$. Analoghi per i coefficienti successivi

Rappresentazione di numeri reali

Esempio: trasformazione del valore 3.125_{10} in base $b = 2$ con $k = 7, f = 4$

valore	base	quoziente	resto
3	2	1	1
1	2	0	1

valore	base	prodotto	parte decimale	parte intera
0.125	2	0.250	.250	0
0.250	2	0.500	.500	0
0.500	2	1.000	.000	1

$$3.125_{10} = 011.0010_2$$

Rappresentazione di numeri reali

Esempio: trasformazione del valore 0.1_{10} in base $b = 2$ con $k = 7, f = 4$

valore	base	prodotto	parte decimale	parte intera
0.1	2	0.2	.2	0
0.2	2	0.4	.4	0
0.4	2	0.8	.8	0
0.8	2	1.6	.6	1

- Ho esaurito le cifre a disposizione: il risultato è 000.0001
- L'ultima parte decimale è diversa da 0: la rappresentazione che ottengo è approssimata

Rappresentazione di numeri reali

Esempio: trasformazione del valore 0.1_{10} in base $b = 2$

valore	base	prodotto	parte decimale	parte intera
0.1	2	0.2	.2	0
0.2	2	0.4	.4	0
0.4	2	0.8	.8	0
0.8	2	1.6	.6	1
0.6	2	1.2	.2	1
0.2	2	0.4	.4	0
0.4	2	0.8	.8	0
0.8	2	1.6	.6	1
0.6	2	1.2	.2	1

Python

- *Rappresenta il valore 0.1 su 50 cifre decimali*

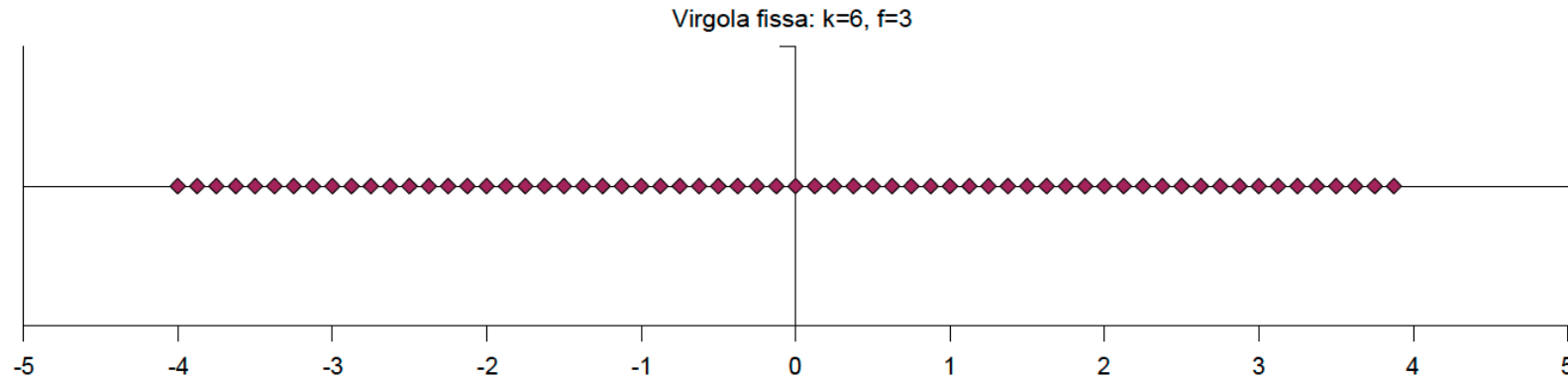
```
>>> print(f'{0.1:.50}')  
0.10000000000000000000000055511151231257827021181583404541
```

Anche in base decimale si verifica lo stesso fenomeno, per altri valori:

$$\frac{1}{2} = 0.5$$

$$\frac{1}{3} = 0.33333 \dots$$

Rappresentazione di numeri reali



- In un sistema di calcolo si può operare soltanto su un numero finito di cifre
- Ha senso utilizzare la stessa *precisione* (errore massimo in valore assoluto) per numeri molto piccoli e numeri molto grandi?
- L'errore massimo dovrebbe essere piccolo per numeri piccoli, e può essere più grande per numeri grandi

Rappresentazione di numeri reali: virgola mobile

- Per numeri grandi o piccoli, diventa preferibile utilizzare la *notazione scientifica*

$$0.00072345_{10} = 0.72345 \cdot 10^{-3}$$

$$723450000_{10} = 0.72345 \cdot 10^9$$

- Vale lo stesso anche in altre basi
- In generale un numero è rappresentato in base b come:

$$(-1)^s \cdot m \cdot b^e$$

- s : segno
- m : mantissa
- e : esponente

Rappresentazione di numeri reali

- Per la rappresentazione in virgola mobile (*floating point*) si utilizzano:
 - 1 bit per il segno
 - k_m bit per la *mantissa*
 - k_e bit per l'*esponente*

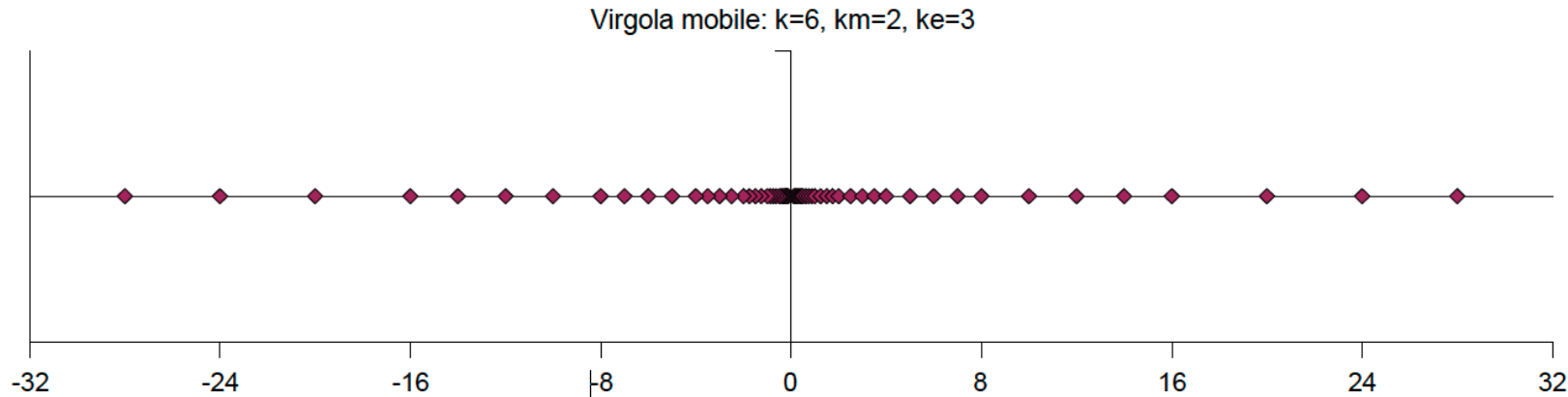
Rappresentazione di numeri reali

Esempio: trasformazione del valore 13.75_{10} in base $b = 2$ con $k = 16$,
 $k_e = 7$ e $k_m = 8$

- Bit di segno: 0
- Conversione in binario della parte intera: $13_{10} = 1101_2$
- Conversione in binario della parte frazionaria: $0.75_{10} = 0.11_2$
- Normalizzazione della mantissa: $1101.11_2 = 1.10111_2 \cdot 2^3$
- Calcolo esponente (*bias pari a $2^{k_e-1} - 1 = 63$*): $66_{10} = 1000010_2$

\pm	Esponente							Mantissa							
0	1	0	0	0	0	1	0	1	0	1	1	1	0	0	0

Rappresentazione di numeri reali : virgola mobile



- Nella rappresentazione in virgola mobile i valori rappresentabili non sono equidistanti
- La *precisione* della codifica dipende dal modulo del valore rappresentato

Rappresentazione di numeri reali

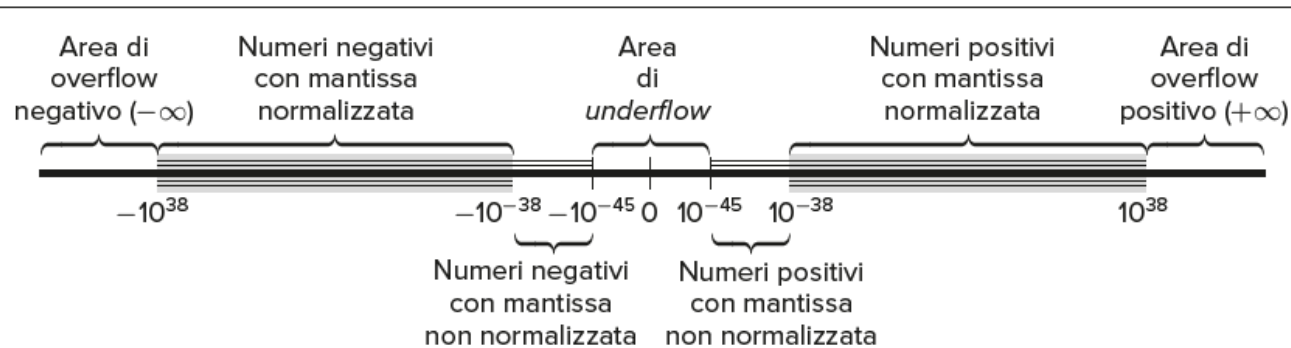
- Per la rappresentazione in virgola mobile (*floating point*) si utilizzano:
 - 1 bit per il segno
 - k_m bit per la *mantissa*
 - k_e bit per l'*esponente*
- La *IEEE* (*Institute of Electrical and Electronic Engineers*) ha definito uno standard per la rappresentazione in virgola mobile dei reali nei sistemi di calcolo

IEEE 754	Segno	k_e (esponente)	k_m (mantissa)
Precisione singola (32 bit)	1	8	23
Precisione doppia (64 bit)	1	11	52

Rappresentazione di numeri reali

Significato	Segno	Valore esponente	Valore mantissa
0	0/1	0	0
$\pm\infty$	0/1	Tutti 1	0
NaN (not a number)	0/1	Tutti 1	$\neq 0$

- Considerando le configurazioni speciali, gli intervalli di rappresentabilità variano:



Precisione	Esponente	Valore	Valore
singola	-127	2^{-127}	$\approx 10^{-38}$
singola	127	2^{127}	$\approx 10^{38}$
doppia	-1023	2^{-1023}	$\approx 10^{-308}$
doppia	+1023	2^{1023}	$\approx 10^{308}$