Università degli Studi di Padova
Corso di Laurea in Matematica
A.A. 2008-2009

INTRODUZIONE ALLA PROGRAMMAZIONE

Barbara Di Camillo

Si ringraziano il Dott. Enrico Grisan e il Dott. Fabio Aiolli per il materiale didattico fornito

Parte II

Rappresentazione dei Dati

Codifica Binaria

I computer hanno una memoria finita. Quindi, l'insieme dei numeri interi e reali che si possono rappresentare in un computer è necessariamente finito

Codifica Binaria

Tutti i dati usati dagli elaboratori sono in forma codificata

Tutti basati soltanto su due cifre 0 e 1 (bit)

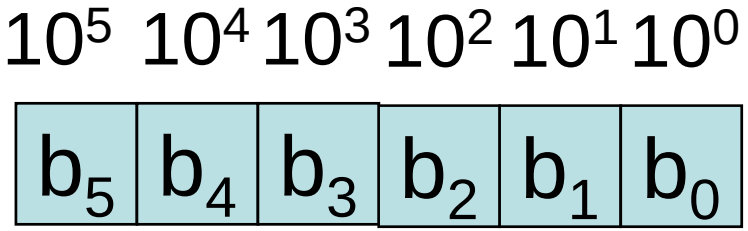
- Perche? Gli strumenti di elaborazione e memorizzazione a cui un calcolatore ha accesso hanno solo DUE stati
  - Schede perforate (Fori in determinati punti o no)
  - Transistors (Conduttivi o no)
  - Nastri Magnetici (Magnetizzati in un verso o un altro)
- Quindi.. Non è necessario un alto grado di precisione (sistemi robusti)

Numeri Interi Positivi

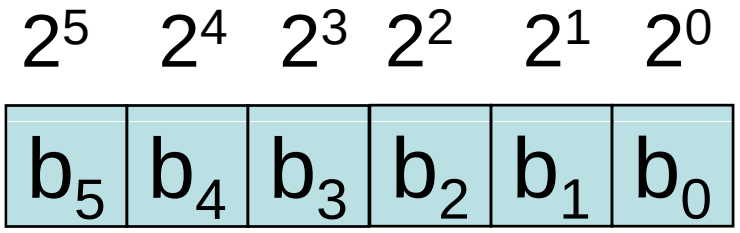
Valore Posizionale

- Il valore di ogni cifra dipende dalla sua posizione nel numero
  - Unità,decine,centinaia.. Nei numeri decimali
  - 1,2,4,8,.. Nei numeri Binari
- La cifra più (meno) significativa è la cifra con il valore posizionale più alto (basso)

Valore Posizionale



$$x = \sum_{k=0}^{6-1} b_k 10^k$$



$$x = \sum_{k=0}^{6-1} b_k 2^k$$

Valore Posizionale

Sia b la base della rappresentazione (2 in binario, 10 in decimale, ecc.)

La cifra in posizione k (da destra verso sinistra) vale b^{k-1}

Sia n il numero di cifre a disposizione. Possiamo codificare b^n numeri

Numeri e Basi

DECIMALI (base 10)

$(134)_{10} = 1\times10^2 + 3\times10^1 + 4\times10^0$

BINARI (base 2)

$(101)_2 = 1\times2^2 + 0\times2^1 + 1\times2^0 = (5)_{10}$

OTTALE (base 8)

$(647)_8 = 6\times8^2 + 4\times8^1 + 7\times8^0 = (423)_{10}$

ESADECIMALE (base 16)

$(123)_{16} = 1\times16^2 + 2\times16^1 + 3\times16^0 = (291)_{10}$

Numeri Ottali e Esadecimali

- I numeri binari sono molto lunghi rispetto alla quantità di info che rappresentano
- Le rappresentazioni ottali e esadecimali sono versioni compatte di numeri binari

OTTALE (a tre a tre)

$(1\ 100\ 111\ 010)_2 = (1472)_8$

ESADECIMALE (a quattro a quattro)

$(11\ 0011\ 1010)_2 = (33A)_{16}$

N.B. Le cifre da 10 a 15 si rappresentano con le lettere A,…,F

Somma di due numeri

- La somma di due numeri (in qualunque base, per un numero fissato di cifre) si può calcolare x colonne

decimale

Riporto	0	1	0		
Addendo 1		5	7	2	+
Addendo 2		1	4	1	=
Somma		7	1	3	

binaria

Riporto	.1	1	1		
Addendo 1		1	0	1	+
Addendo 2		0	1	1	=
Somma		0	0	0	

Se ho solo 3 cifre a disposizione in numerazione binaria si è verificato un OVERFLOW

Da decimale a Binario

Numero	Parte Intera/2	Resto
142	71	0
71	35	1
35	17	1
17	8	1
8	4	0
4	2	0
2	1	0
1	0	1

$(142)_{10}$ =

$(10001110)_2$

$(216)_8$

$(8E)_{16}$

Rappresentazione degli interi

Generalmente (dipende dalla macchina e dal contesto d'uso) un intero viene rappresentato in 4 byte = 32 bit

Quindi si possono rappresentare 2^{32} (circa 4 miliardi e 300 milioni) interi diversi

Si potrebbero quindi rappresentare tutti gli interi non negativi nell'intervallo $[0, 2^{32}-1]$

E i negativi?

Numeri Interi

Interi Negativi

Vedremo tre tipi di codifica per i negativi

1. Bit e segno
2. Complemento a uno
3. Complemento a due

Bit e Segno

Riserviamo il primo bit per il segno:
0 = positivo
1 = negativo.

I numeri non negativi rappresentabili sono quindi quelli rappresentabili con n-1 bit, cioè nell'intervallo $[0, 2^{n-1}-1]$

s(+/-)	2^4	2^3	2^2	2^1	2^0
b_5	b_4	b_3	b_2	b_1	b_0

$$x = s\sum_{k=0}^{n-2} b_k \cdot 2^k$$

Bit e Segno

Utilizzando la codifica "Bit e Segno", non possiamo più eseguire agevolmente la somma di due numeri per colonne.

binaria

Riporto	0	0	1		
Addendo 1		1	0	1	+
Addendo 2		0	0	1	=
Somma		1	1	0	

$(-1_{10})+$
$(1_{10})=$
(-2_{10})

La somma di 101 e 001 dovrebbe darmi 000, non 110 !!!

Inoltre ci sono due rappresentazioni dello 0! Ad esempio, con 3 cifre: 100 e 000.

Complemento a uno

Riserviamo il primo bit per il segno:
0 = positivo
1 = negativo.

I numeri non negativi rappresentabili sono quindi quelli rappresentabili con n-1 bit, cioè nell'intervallo $[0,2^{n-1}-1]$ e ottenibili complementando il numero a (2^n-1).

$(2^n-1)=(2^6-1)=63$	1	1	1	1	1	1	-
$20=2^2+2^4$	0	1	0	1	0	0	=
Rappresentazione di -20	1	0	1	0	1	1	(43_{10})

Complemento a uno

Anche in questo caso ci sono due rappresentazioni dello 0! Ad esempio, con 6 cifre: 000000 e 111111.

$(2^n-1)=(2^6-1)=63$	1	1	1	1	1	1	-
0	0	0	0	0	0	0	=
Rappresentazione di -0	1	1	1	1	1	1	

Complemento a uno

Ma possiamo eseguire agevolmente la somma di due numeri per colonne.

binaria

Riporto	0	0	0		
Addendo 1		1	1	0	+
Addendo 2		0	0	1	=
Somma		1	1	1	

$(-1_{10})+$
$(1_{10})=$
(0_{10})

Complemento a due

Voglio usare una rappresentazione dei numeri negativi, magari meno intuitiva, ma che

- mi consenta di eseguire comodamente le operazioni
- Abbia una rappresentazione univoca dello 0

Complemento a due

Riserviamo il primo bit per il segno:
0 = positivo
1 = negativo.

I numeri non negativi rappresentabili sono quindi quelli rappresentabili con n-1 bit, cioè nell'intervallo $[0,2^{n-1}-1]$ e ottenibili complementando il numero a (2^n).

$(2^n)=(2^6)=64$

$20=2^2+2^4$

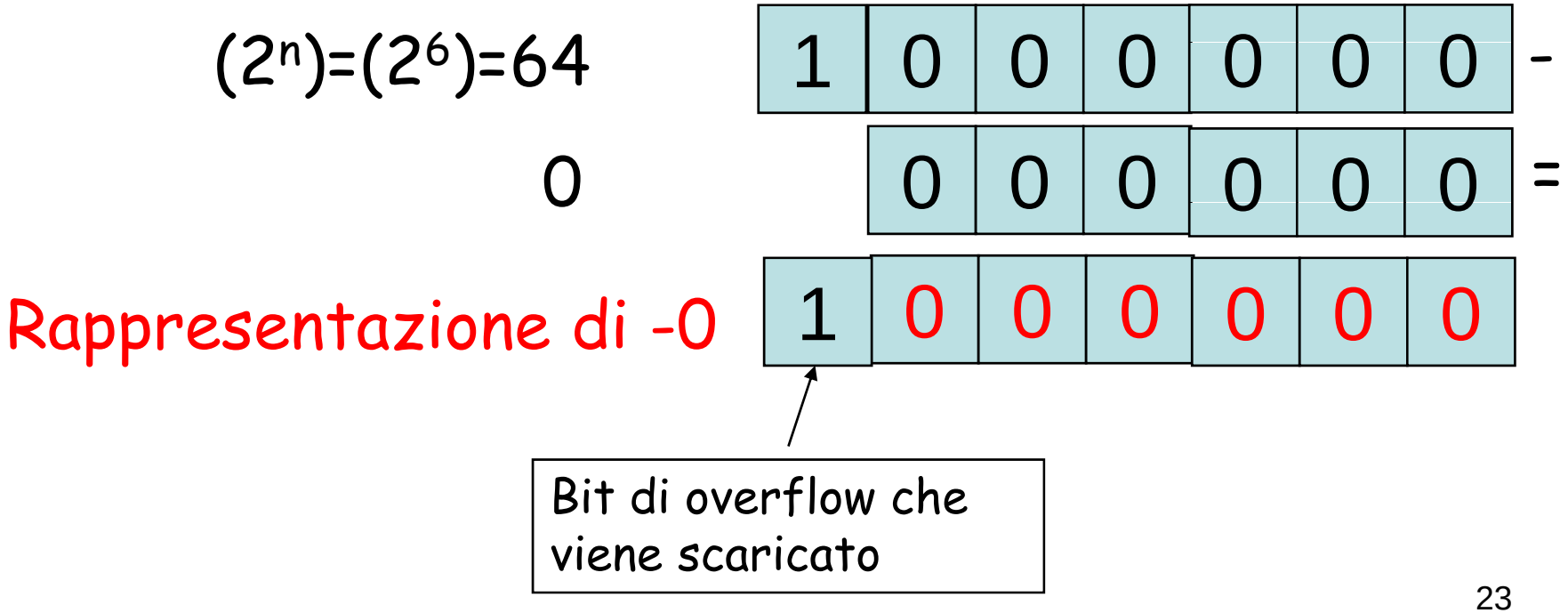
	1	0	0	0	0	0	0
-		0	1	0	1	0	0
=	0	1	0	1	1	0	0

Rappresentazione di -20

Bit di overflow che viene scaricato

Complemento a due

C'è una sola rappresentazione dello 0:
000000



Complemento a due

E possiamo eseguire agevolmente la somma di due numeri per colonne.

binaria

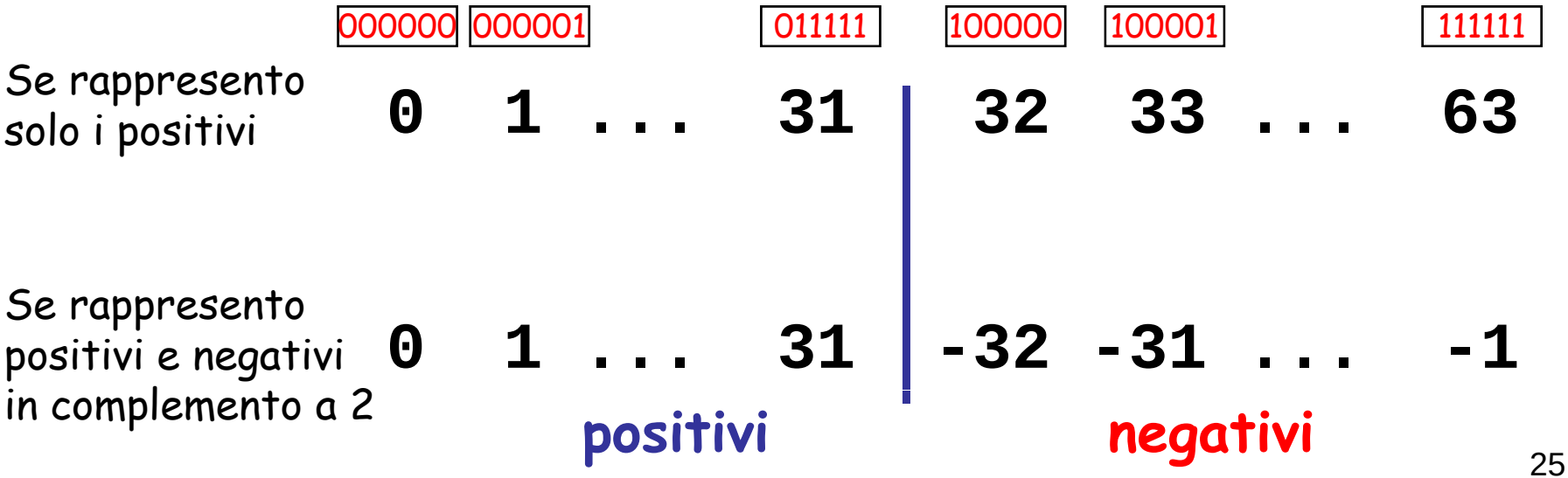
Riporto	1	1	1		
Addendo 1		1	1	1	+
Addendo 2		0	0	1	=
Somma		0	0	0	

$(-1_{10})+$
$(1_{10})=$
(-2_{10})

La somma di 101 e 001 mi da 000!

Complemento a due

Per ottenere un numero in complemento a due, posso anche considerare i valori posizionali x un intero a 6 bit sono: -32 16 8 4 2 1



Complemento a due

Per un intero positivo a 6 bit (senza contemplare la rappresentazione di numeri negativi):

2^5 2^4 2^3 2^2 2^1 2^0

b_5	b_4	b_3	b_2	b_1	b_0

$$x = \sum_{k=-2}^{3-1} b_k 2^k$$

Per un intero a 6 bit (rappresentazione in complemento a due):

-2^5 2^4 2^3 2^2 2^1 2^0

b_5	b_4	b_3	b_2	b_1	b_0

$$x = b_{6-1}\left(-2^{6-1}\right) + \sum_{k=0}^{6-2} b_k 2^k$$

Complemento a due (cambio di segno)

Con la rappresentazione in complemento a due possiamo invertire un numero invertendo i bit e sommando 1.

Inversione bit + 1

19 in complemento a 2	010011	+19
Inverti i bit	101100	↓
Somma 1	101101	-19

-23 in complemento a 2	101001	-23
Inverti i bit	010110	↓
Somma 1	010111	+23

Complemento a due (sottrazione)

Con la rappresentazione in complemento a due possiamo eseguire la sottrazione sommando al minuendo il sottraendo cambiato di segno.

12-14=12+(-14)

Memorizza 14	001110
Inverti i bit	110001
Somma 1, ottieni -14	110010
Memorizza 12	001100
Somma -14 e 12	111110

-2

Complemento a due (overflow)

Con la rappresentazione in complemento a due è facile riconoscere l'overflow.

Esempio 1: 14+9

		-32	16	8	4	2	1	
Riporto	0	0	1	0	0	0		
14		0	0	1	1	1	0	+
9		0	0	1	0	0	1	=
Somma		0	1	0	1	1	1	23

OK

Complemento a due (overflow)

Con la rappresentazione in complemento a due è facile riconoscere l'overflow.

Esempio 2: 25+18

		-32	16	8	4	2	1	
Riporto	0	1	0	0	0	0		
25		0	1	1	0	0	1	+
18		0	1	0	0	1	0	=
Somma		1	0	1	0	1	1	

-21

NO

Complemento a due (overflow)

Con la rappresentazione in complemento a due è facile riconoscere l'overflow.

Esempio 3: 17+(-13)

		-32	16	8	4	2	1	
Riporto	1	1	0	0	1	1		
17		0	1	0	0	0	1	+
-13		1	1	0	0	1	1	=
Somma		0	0	0	1	0	0	

4

OK

Complemento a due (overflow)

Con la rappresentazione in complemento a due è facile riconoscere l'overflow.

Esempio 4: (-7)+(-31)

		-32	16	8	4	2	1	
Riporto	1	0	0	0	0	1		
-7		1	1	1	0	0	1	+
-31		1	0	0	0	0	1	=
Somma		0	1	1	0	1	0	

26

NO

Errore di Overflow

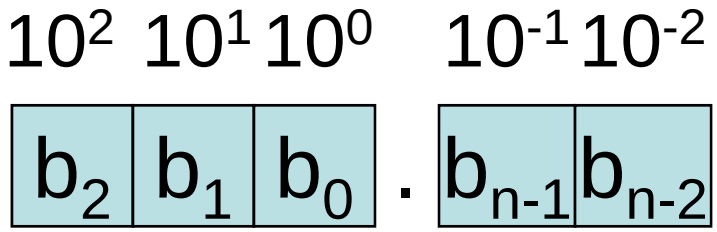
Si può dimostrare che una somma di due numeri di n cifre in complemento a 2 dà (errore di) overflow **se e solo se** i riporti in colonna *n* e *n* +1 sono diversi

Numeri Reali

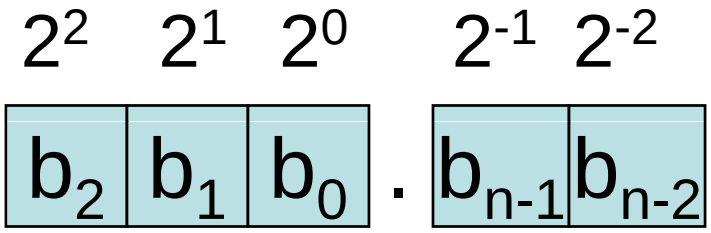
Valore Posizionale

- Il valore di ogni cifra dipende dalla sua posizione nel numero
  - Unità,decine,centinaia.. Nei numeri decimali
  - 1,2,4,8,.. Nei numeri Binari
  - Decimi,centesimi.. Nelle frazioni decimali
  - Metà, quarti.. Nelle frazioni binarie
- La cifra più (meno) significativa è la cifra con il valore posizionale più alto (basso)

Valore Posizionale



$$x = \sum_{k=-2}^{3-1} b_k 10^k$$

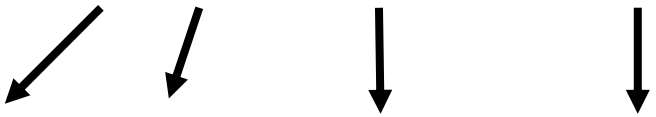


$$x = \sum_{k=-2}^{3-1} b_k 2^k$$

Numeri decimali

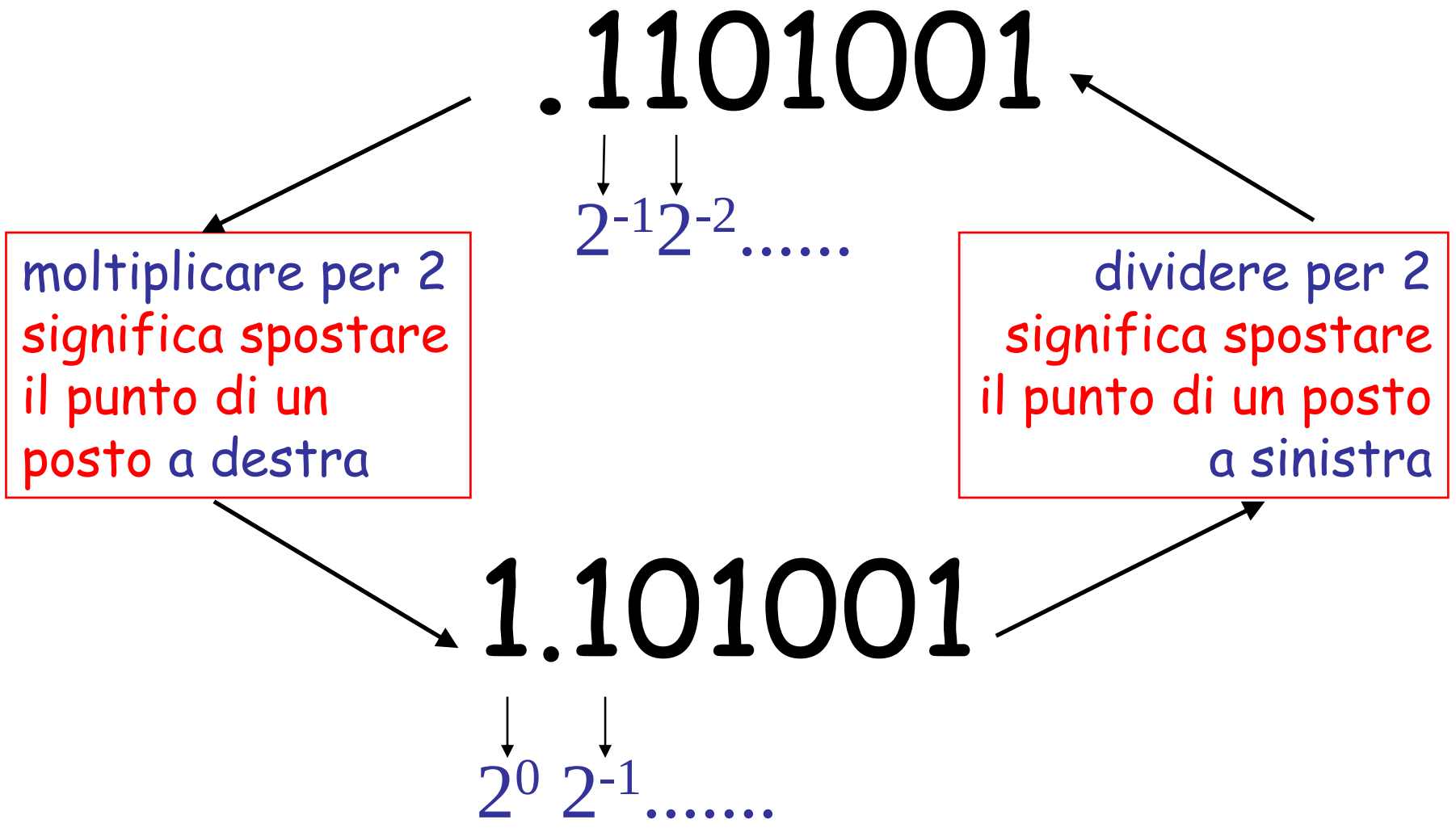
Cosa significa una parte decimale binaria?

.1101001



$2^{-1} + 2^{-2} + 2^{-4} + 2^{-7} = .5+.025+.0125+.00625$

Numeri decimali



Numeri decimali

- Il trucchetto di spostare la virgola x moltiplicare o dividere un numero funziona anche per le altre basi !
- Esempi:
  - $(1.234)_{10} / 10 = (.1234)_{10}$
  - $(.01234)_{10} \times 100 = (1.234)_{10}$
  - Ma anche $(463.427)_8 / 64 = (4.63427)_8$

  E così via..

Numeri decimali

Se abbiamo un valore decimale in base 10, ad esempio 0.99, come troviamo la sua rappresentazione in base 2? Ragioniamo come segue:

Supponiamo che

$(.99)_{10} = .b_1b_2b_3...b_k$ (in binario)

allora $2 \times .99 = 1.98 = \mathbf{b_1}.b_2...b_k$

quindi b_1 è 1

e .98 è rappresentato da $.b_2...b_k$

Quindi: per trovare la rappresentazione binaria di un valore decimale lo moltiplichiamo per 2 e osserviamo la cifra che appare nella parte intera.

Esempio: $(.59)_{10} = (?)_2$

.59×2= 1.18

.18×2= 0.36

.36×2= 0.72

.72×2= 1.44

.44x2= 0.88

.88x2= 1.76

…

$(0.59)_{10}=(.100101\ldots)_2$

E così via… dipende da quanti bit abbiamo a disposizione!

N.B. Mi posso fermare se moltiplicando per 2 ottengo una parte decimale = 0

Esempio Completo

$(18.59)_{10} = (\ ?\ )_2$

$(18)_{10} = (10010)_2$

$(.59)_{10} = (.100101\ldots)_2$

Ne deriva che:

$(18.59)_{10} = (10010.100101\ldots)_2$

Rappresentazione in virgola mobile

Osserviamo che spostando la virgola di h posti verso destra nella rappresentazione decimale di un numero N moltiplichiamo N per 10^h mentre spostandola verso sinistra di h posti N viene diviso per 10^h. Ad esempio:

$18.59 = 18.59 \times 10^0$
$= 1.859 \times 10^1$ rappresentazione normalizzata
$= 0.1859 \times 10^2$
$= 185.9 \times 10^{-1}$
$= 1859 \times 10^{-2}$

Si parla di rappresentazione in virgola mobile. Quindi in virgola mobile lo stesso numero decimale ha piu' rappresentazioni possibili. Solitamente si usa la rappresentazione normalizzata in cui la parte intera ha un'unica cifra decimale diversa da zero.

Rappresentazione in virgola mobile

La rappresentazione in virgola mobile per i numeri binari e` del tutto analoga. Ad esempio:

$$
\begin{aligned}
101.01 &= 101.01 \times 2^{0} \\
&= 10.101 \times 2^{1} \\
&= 1.0101 \times 2^{2} \\
&= 1010.1 \times 2^{-1} \\
&= 10101 \times 2^{-2}
\end{aligned}
$$

Rappresentazione normalizzata

La rappresentazione in virgola mobile normalizzata ha <u>sempre</u> parte intera uguale a 1.

Per registrare un numero reale x in memoria si adotta la rappresentazione binaria in virgola mobile normalizzata:

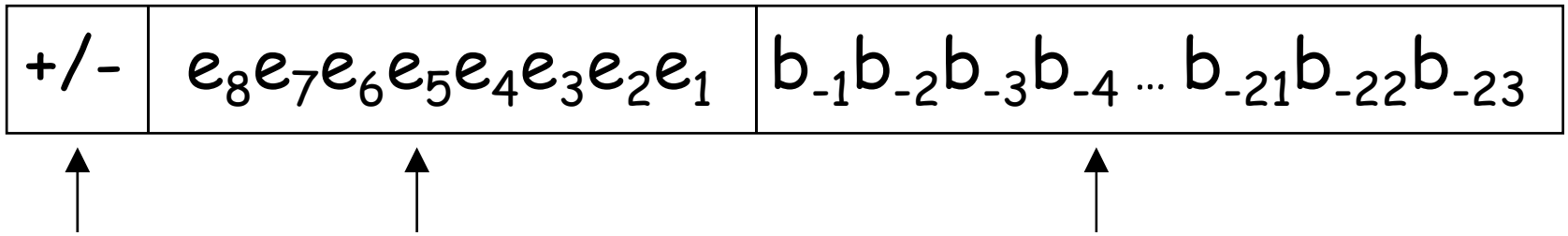
$$x = s(+/-)\ 2^{e} \times 1.b_{-1}b_{-2}b_{-3}b_{-4}\ldots$$

Naturalmente non e` possibile memorizzare la sequenza possibilmente infinita di bit della parte frazionaria ma si memorizzano soltanto i primi bit. Anche per l'esponente si utilizzano un numero finito di bit. Generalmente per rappresentare un numero reale si usano 4 byte nel modo seguente:

+/-	$e_8e_7e_6e_5\ e_4e_3e_2e_1$	$b_{-1}b_{-2}b_{-3}b_{-4}\ldots b_{-21}b_{-22}b_{-23}$
1 bit	**8 bit**	**23 bit**
Segno	**Esponente**	**Mantissa**

Rappresentazione in virgola mobile

$$x = s(+/-)\ 2^{e} \times 1.b_{-1}b_{-2}b_{-3}b_{-4}\ldots$$



- Il primo bit rappresenta il segno del numero: 0 per +, 1 per -.
- Gli otto bit successivi si usano per rappresentare l'esponente.
- I ventitre bit successivi servono a rappresentare la parte decimale del numero (rappresentazione normalizzata)

Rappresentazione in virgola mobile

ESPONENTE

- Gli otto bit dell'esponente sono numeri interi con segno (l'esponente può avere segno positivo onegativo), perciò è possibile utilizzare le diverse notazioni per rappresentarli (bit e segno, complemento a uno, complemento a due).
- Tuttavia, per consentire un confronto più agevole dei numeri reali, lo standard attualmente utilizzato è lo IEEE 754, ufficialmente: IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) o anche IEC 60559:1989, Binary floating-point arithmetic for microprocessor systems).
- Lo standard prevede di rappresentare l'esponente come un intero positivo (escludendo le sequenze di bit 00000000 e 11111111, che sono riservate) e di sotrarre 127 all'intero positivo rappresentato. **Esponente $e = e_8e_7e_6e_5e_4e_3e_2e_1 - 127$**
- Gli interi rappresentabili sono allora contenuti nel range: (00000001-01111111)=(1-127)=-126 e (11111110-01111111)=(254-127)=127.

Rappresentazione in virgola mobile

Quindi si rappresenta il numero

$$N = s\ 2^{e^*-127} \times 1.m$$

Rappresentazione in virgola mobile

0	10000011	00111001011000000000000

10000011 = 131, quindi esponente = 131-127 = 4.
Quindi

$2^4 \times 1.00111001011 = 10011.1001011$

$= 2^4 + 2^1 + 2^0 + 2^{-1} + 2^{-4} + 2^{-6} + 2^{-7}$

$= 19.5859375$

Esercizio: Fornire la rappresentazione in virgola mobile normalizzata del valore 10.543 avendo a disposizione 8 bit per l'esponente e 8 per la mantissa.

(1) Rappresentiamo 10 in binario

$10 = 2^3 + 2^1 = (1010)_2$

(2) Rappresentiamo 0.543 in binario

$0.543 \times 2 = 1.086$

$0.086 \times 2 = 0.172$

$0.172 \times 2 = 0.344$ Quindi: $0.543 = (0.100010\ldots)_2$

$0.344 \times 2 = 0.688$

$0.688 \times 2 = 1.376$

$0.376 \times 2 = 0.752$

(3) Riassumendo:

$10.543 = (1010.100010...)_2$

(4) Rappresentazione normalizzata

$1.01010001 \times 2^3 = 1010.10001$

(5) Rappresentiamo l'esponente 3:

$e=3=e^*-127$, quindi $e^*=130$

$130 = (10000010)_2$

QUINDI:

0	**10000010**	**01010001**

Rappresentazione in virgola mobile

Cosa viene rappresentato nel campo esponente con le sequenze di bit 00000000 (0) e 11111111 (255)?

Categoria	Esp.	Mantissa
Zeri	0	0
Numeri subnormalizzati	0	non zero
Numeri normalizzati	1-254	qualunque
Infiniti	255	0
Nan (not a number)	255	non zero

I numeri subnormalizzati sono i numeri reali "piccoli", ovvero minori in valore assoluto del più piccolo numero rappresentabile utilizzando la notazione normalizzata:

Per un numero normalizzato: $N = s\ 2^{e^*-127} \times 1.m$

Per un numero denormalizzato: $N = s\ 2^{-126} \times 0.m$

Rappresentazione dei Reali
"piccoli"

Rappresenta:

$$x = s\ 2^{-126}\ 0,m$$

implicito

$0 \leq 0,m < 1$
quindi
$-2^{-126} < x < 2^{-126}$

Rappresentazione di Infinito

Rappresenta: $\pm\infty$

Si può ottenere come risultato di qualche operazione aritmetica (es: divisione per 0 di un numero diverso da 0).

Rappresentazione di NaN

Rappresenta: **NaN**

Si può ottenere come risultato di qualche operazione aritmetica (es: divisione di 0 per 0).

Quanti decimali si rappresentano?

Con 32 bit possiamo rappresentare al piu' 2^{32} valori distinti. Questi valori però non sono distribuiti uniformemente come gli interi, bensì sono maggiormente concentrati tra -1 e 1 e si diradano sempre più allontanandosi dallo 0

Distribuzione disuniforme

h=2 bits di mantissa e k=3 di esponente.

Rappresentazione dell'esponente:

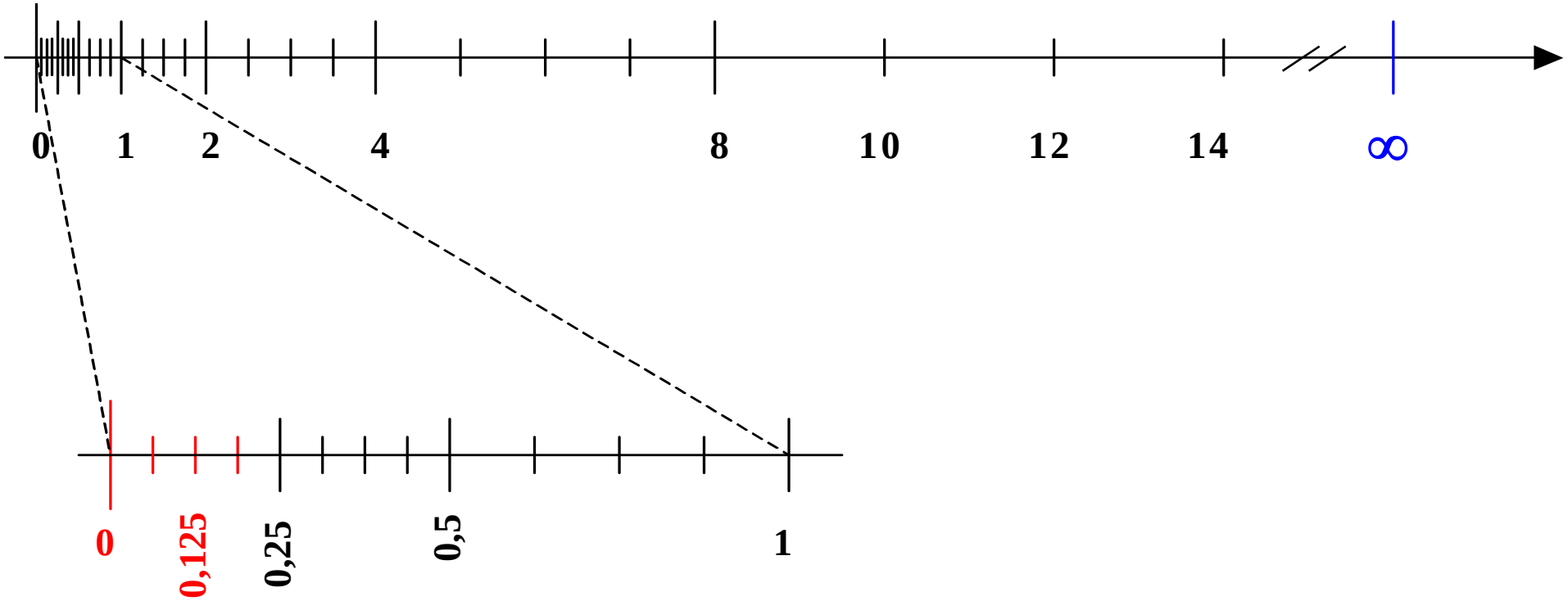
$$e = e^* - \left(2^{k-1} - 1\right) = e^* - 3$$

0 000 00 =	0	= 0
0 000 01 =	$2^{-2} \cdot 2^{-2}$	= 0.0625
0 000 10 =	$2^{-2} \cdot 2^{-1}$	= 0,125
0 000 11 =	$2^{-2} \cdot (2^{-1} + 2^{-2})$	= 0,1875
0 001 00 =	$2^{-2} \cdot 1$	= 0,25
0 001 01 =	$2^{-2} \cdot (1 + 2^{-2})$	= 0,3125
0 001 10 =	$2^{-2} \cdot (1 + 2^{-1})$	= 0,375
0 001 11 =	$2^{-2} \cdot (1 + 2^{-1} + 2^{-2})$	= 0,4375
0 010 00 =	$2^{-1} \cdot 1$	= 0,5
0 010 01 =	$2^{-1} \cdot (1 + 2^{-2})$	= 0,625
0 010 10 =	$2^{-1} \cdot (1 + 2^{-1})$	= 0,75
0 010 11 =	$2^{-1} \cdot (1 + 2^{-1} + 2^{-2})$	= 0,875

Distribuzione disuniforme cont.

0 011 00 =	$2^0 \cdot 1$	= 1
0 011 01 =	$2^0 \cdot (1+2^{-2})$	= 1,25
0 011 10 =	$2^0 \cdot (1+2^{-1})$	= 1,5
0 011 11 =	$2^0 \cdot (1+2^{-1}+2^{-2})$	= 1,75
0 100 00 =	$2^1 \cdot 1$	= 2
0 100 01 =	$2^1 \cdot (1+2^{-2})$	= 2,5
0 100 10 =	$2^1 \cdot (1+2^{-1})$	= 3
0 100 11 =	$2^1 \cdot (1+2^{-1}+2^{-2})$	= 3,5
0 101 00 =	$2^2 \cdot 1 = 4$	= 4
0 101 01 =	$2^2 \cdot (1+2^{-2})$	= 5
0 101 10 =	$2^2 \cdot (1+2^{-1})$	= 6
0 101 11 =	$2^2 \cdot (1+2^{-1}+2^{-2})$	= 7
0 110 00 =	$2^3 \cdot 1$	= 8
0 110 01 =	$2^3 \cdot (1+2^{-2})$	= 10
0 110 10 =	$2^3 \cdot (1+2^{-1})$	= 12
0 110 11 =	$2^3 \cdot (1+2^{-1}+2^{-2})$	= 14
0 111 00 =	∞	= ∞

Distribuzione disuniforme



Caratteri

In generale viene usata la **codifica standard ASCII**:

ogni carattere è rappresentato in 1 byte e quindi possiamo rappresentare 256 caratteri. Questo basta per:

a...z A...Z 0...9 . , ;: () etc

+ caratteri di controllo: Enter, Tab, etc

Tabella ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

Tabella ASCII

128	Ç	144	É	161	í	177	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	163	ú	179	│	195	├	211	╙	227	π	243	≤
131	â	147	ô	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
134	å	150	û	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
136	ê	152	ÿ	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
137	ë	153	Ö	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	171	½	187	╗	203	╦	219	█	235	δ	251	√
139	ï	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
140	î	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
141	ì	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	
143	Å	160	á	176	░	192	└	208	╨	224	α	240	≡		

Source: www.asciitable.com

Wide characters

Nel caso sia necessario rappresentare più caratteri, per esempio caratteri usati in lingue asiatiche, esiste la **codifica UNICODE** che associa 2 byte ad ogni carattere. In questo modo si possono rappresentare 65536 caratteri diversi

Qualche esercizio (tipo esame..)

1. Per il numero $(147)_{10}$ dare la versione binaria, ottale e esadecimale. E in base 7 come sarebbe?
2. Eseguire la somma $(234)_{10}+(145)_{10}$, nelle tre basi binaria, ottale e esadecimale.
3. Usando la rappresentazione in complemento a 2, quali valori interi si possono rappresentare con 5 bit?
4. Supponiamo di rappresentare gli interi in complemento a due su 5 bit. Mostrare le seguenti somme e sottrazioni: -5-8 e 10+8. Vi e` overflow? E il risultato e' corretto?
5. Fornire la rappresentazione in virgola mobile normalizzata dei valori 0.5, 1.5 e 14.64 avendo a disposizione 1 bit per il segno, 8 bit per l'esponente e 8 per la mantissa