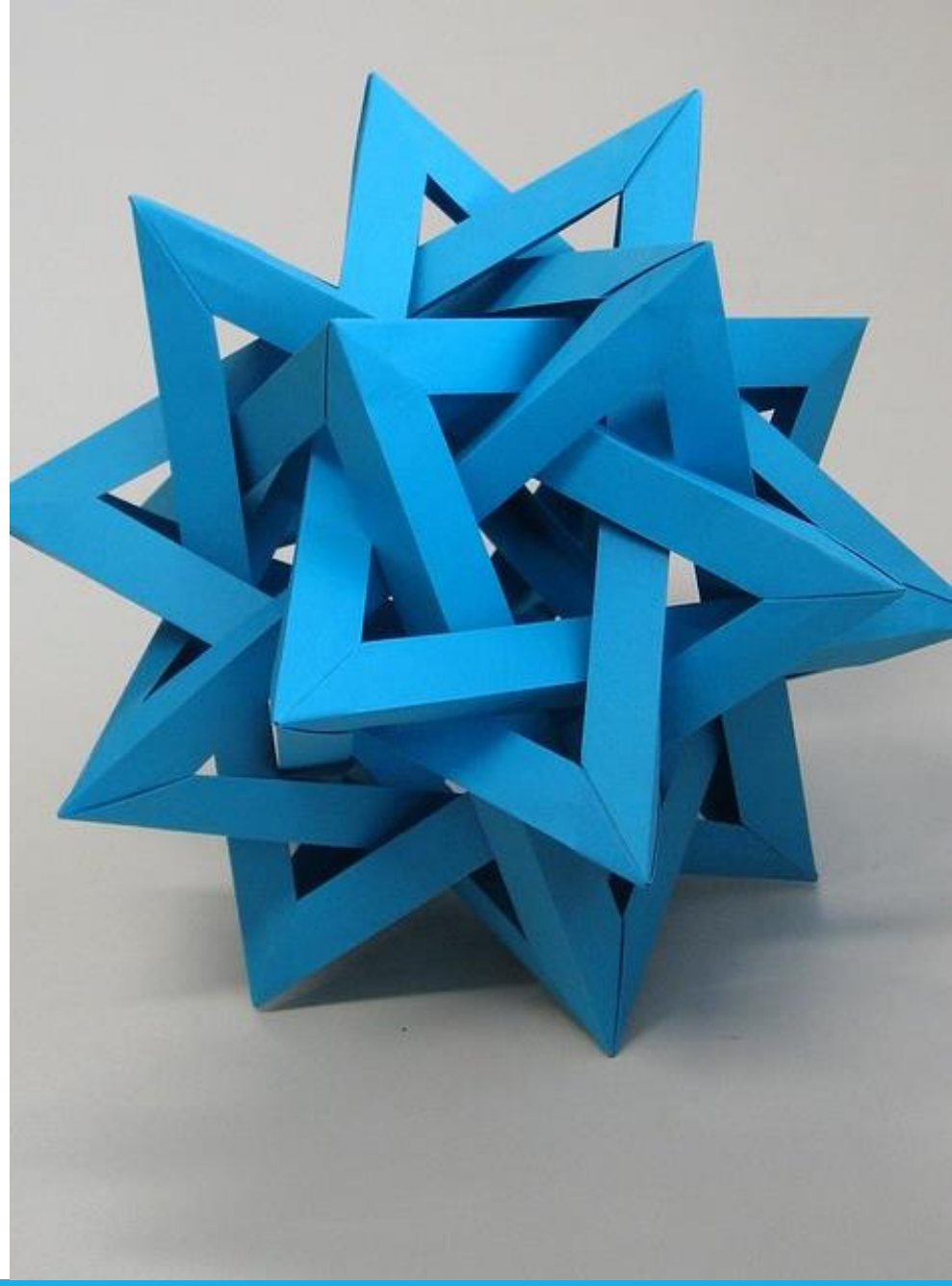
Politecnico di Torino
Dipartimento di Automatica e Informatica

Unità T1: Rappresenta-zione dei dati



Come contiamo?

- Il sistema di numerazione del mondo occidentale (sistema indo-arabo) è:
  - **decimale**
  - **posizionale**

$$252 =$$

Come contiamo?

- Il sistema di numerazione del mondo occidentale (sistema indo-arabo) è:
  - decimale
  - posizionale

$$252 = 2 \times 100 + 5 \times 10 + 2 \times 1$$
$$= 2 \times 10^2 + 5 \times 10^1 + 2 \times 10^0$$

Sistemi di numerazione

- Non posizionali (additivi):
  - egiziano
  - romano
  - greco

- Posizionali:
  - babilonese (2 cifre, sessagesimale)
  - inuit, selti, maya (ventesimale)
  - indo-arabo (decimale)

- Ibridi:
  - cinese

Sistema di numerazione posizionale

- Occorre definire la base B da cui discendono varie caratteristiche:
  - cifre = { 0, 1, 2, ..., B-1 }
  - peso della cifra i-esima = B^i
  - rappresentazione (numeri naturali) su N cifre
    - $a_{N-1}\ a_{N-2} \ldots a_3\ a_2\ a_1\ a_0$

$$A = \sum_{i=0}^{N-1} a_i \cdot B^i$$

Il sistema binario

- Base = 2

- Cifre = { 0, 1 }
- BIT = BInary DigiT

- Esempio:

$$101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 1 \times 4 + 1 \times 1$$
$$= 5_{10}$$

Binario e Decimale

ALCUNI NUMERI BINARI

0 … 0		1000…	8
1 … 1		1001 …9	
10 … 2		1010 …10	
11 … 3		1011 …11	
100 …	4	1100…12	
101 …	5	1101 …13	
110 …	6	1110 …14	
111 …	7	1111 …15	

ALCUNE POTENZE DI DUE

2^0…1		2^9…	512
2^1…2		2^{10}…	1024
2^2…4		2^{11}…	2048
2^3…8		2^{12}…	4096
2^4…16		2^{13}…	8192
2^5…32		2^{14}…	16384
2^6…64		2^{15}…	32768
2^7…128		2^{16}…	65536
2^8…256			

Conversione di numeri naturali da binario a decimale

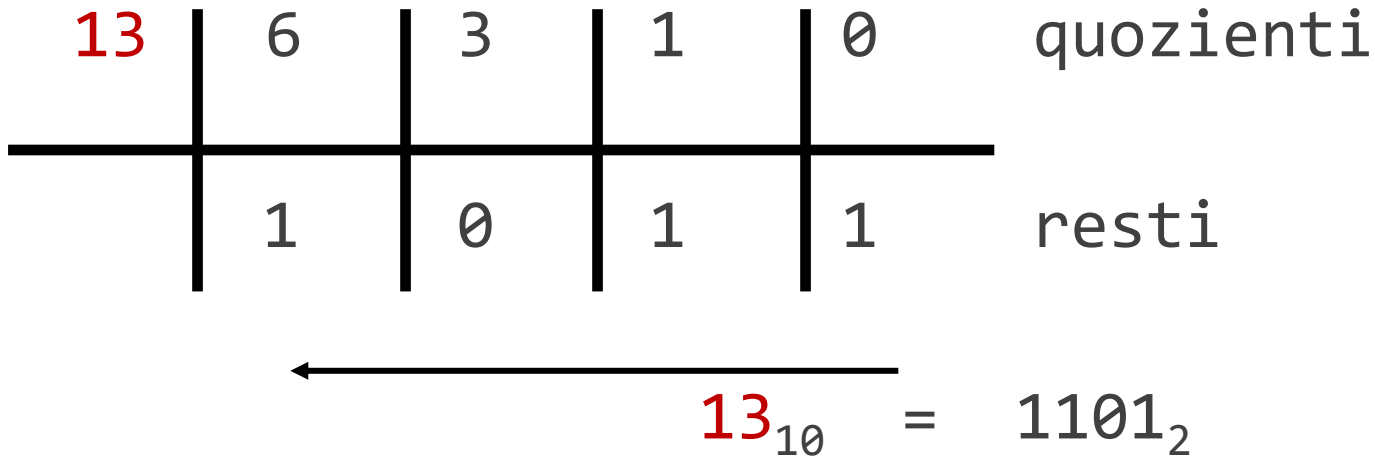
- Si applica direttamente la definizione effettuando la somma pesata delle cifre binarie:

$$
\begin{aligned}
1101_2 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
&= 8 + 4 + 0 + 1 \\
&= 13_{10}
\end{aligned}
$$

Conversione da sistema decimale a binario

- Dall'interpretazione della codifica binaria

- Regola pratica:
  - Divisioni successive per due
  - Si prendono i resti in ordine inverso



Limiti del sistema binario (rappresentazione naturale)

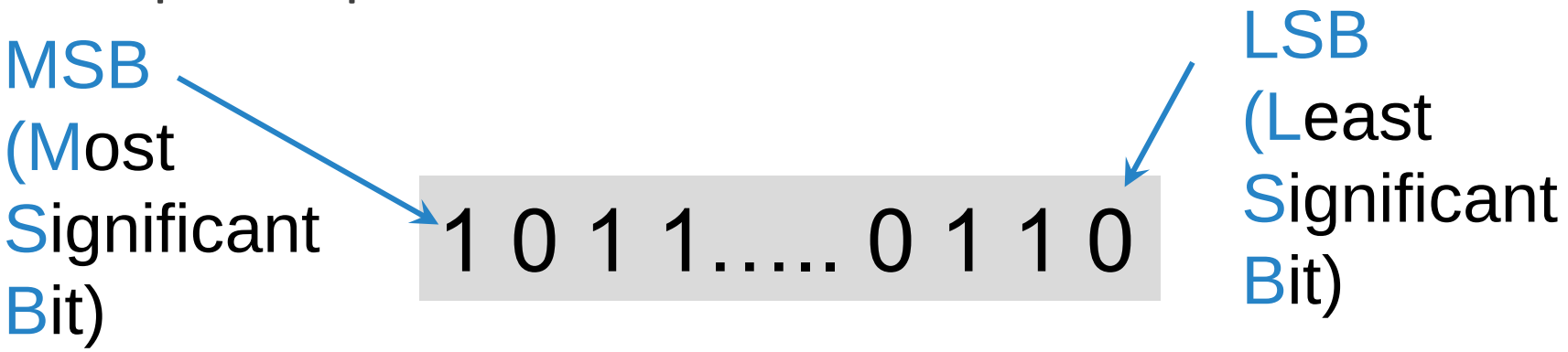
- Consideriamo numeri naturali in binario:
  - 1 bit ~ 2 numeri ~ $\{ 0, 1 \}_2 \sim [0 \dots 1]_{10}$
  - 2 bit ~ 4 numeri ~ $\{ 00, 01, 10, 11 \}_2 \sim [0 \dots 3]_{10}$

- Quindi in generale per numeri naturali a N bit:
  - combinazioni distinte: 2^N
  - intervallo di valori

$$0 \leq x \leq 2^N - 1 \quad [\text{ base 10 }]$$
$$(000\ldots0) \leq x \leq (111\ldots1) \quad [\text{ base 2 }]$$

Terminologia

- Bit rappresenta una singola cifra
- Aggregazioni di bit rilevanti:
  - Byte = 8 bit
- Word = aggregazione di byte
  - 1,2,4,8
  - Utilizzate per le celle di memoria
- Dato un qualunque numero di bit



Limiti del sistema binario (rappresentazione naturale)

Bit	Simboli	Min_{10}	Max_{10}
4	16	0	15
8	256	0	255
16	65 536	0	65 535
32	4 294 967 296	0	4 294 967 295

Somma in binario

- Regole base:

0	+	0	=	0	
0	+	1	=	1	
1	+	0	=	1	
1	+	1	=	0	(carry = 1)

- Si effettuano le somme parziali tra i bit dello stesso peso, propagando gli eventuali riporti:

1	***1***			
0	1	1	0	+
0	1	1	1	=
1	1	0	1	

Sottrazione in binario

- Regole base:

0 - 0 = 0

0 - 1 = 1 (borrow = 1)

1 - 0 = 1

1 - 1 = 0

Sottrazione in binario

- Si effettuano le differenze parziali tra i bit dello stesso peso, gestendo gli eventuali prestiti:

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

Overflow

- Si usa il termine *overflow* per indicare l'errore che si verifica in un sistema di calcolo automatico quando il risultato di un'operazione **non è rappresentabile con la medesima codifica e numero di bit degli operandi**.

- L'overflow è una condizione "dinamica"
  - Esiste solo come risultato di un'operazione

Overflow

- Nella somma in binario puro si ha overflow quando:
  - si lavora con numero fisso di bit
  - si ha carry sul MSB

- Esempio: numeri da 4 bit codificati in binario puro

```
  0101 +
  1110 =
---------
 10011
```

overflow

Overflow e linguaggi di programmazione

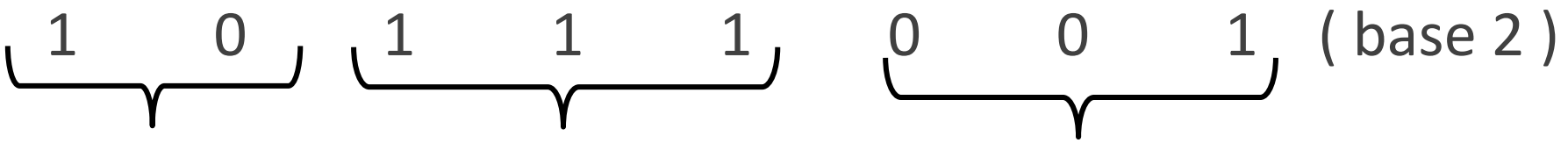
- In generale nei linguaggi di programmazione questi limiti sono reali
- Per esempio in C, dove gli interi occupano in genere 32 bit
  - Il massimo numero positivo rappresentabile è **2^{31}-1 = 2147483647**
  - **Qualsiasi operazione che ecceda questo valore 'riparte da zero'**
    - Esempio:

```
x = 2^31-1
x = x + 1          # fa ZERO !!!
```

- Python usa invece una **rappresentazione degli interi a precisione arbitraria**
  - NON SI HA MAI OVERFLOW TRA INTERI!!!!

Il sistema ottale

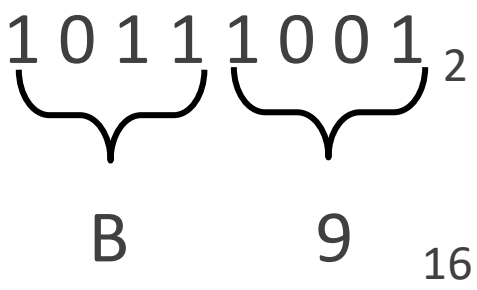
- base = 8 (talvolta indicata con Q per Octal)
  - cifre = { 0, 1, 2, 3, 4, 5, 6, 7 }
  - utile per scrivere in modo compatto i numeri binari (3:1)



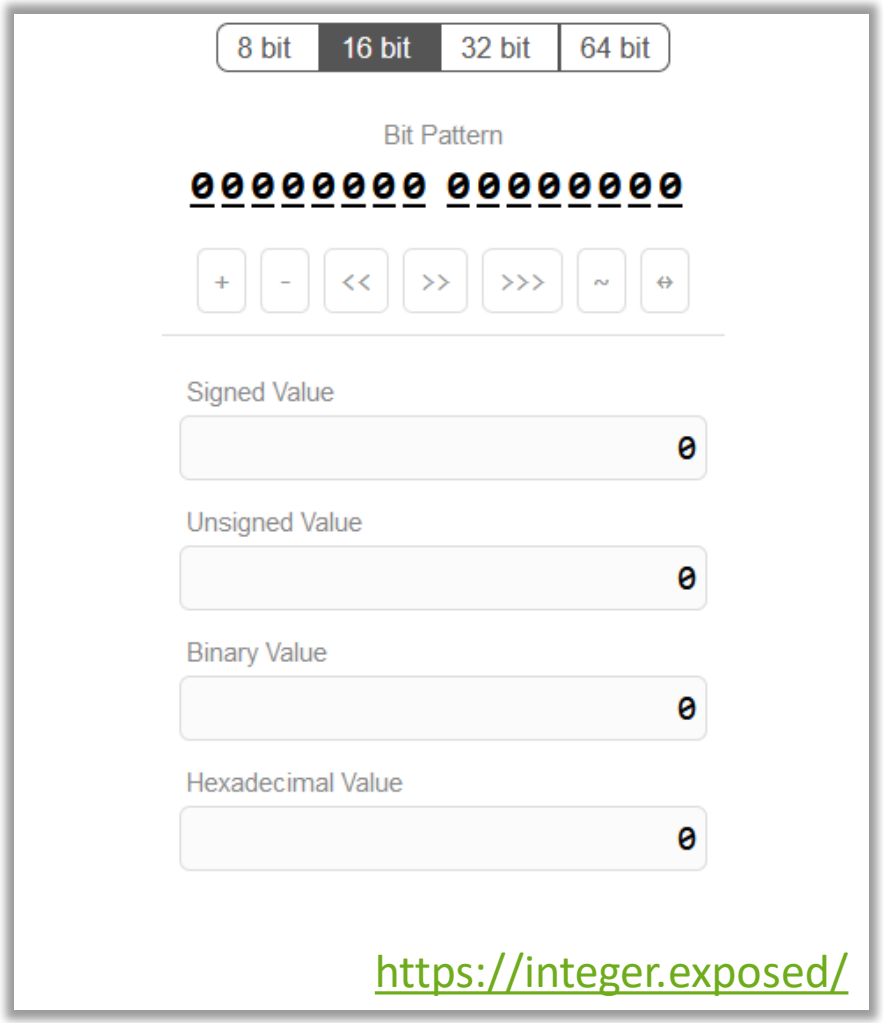
2 7 1 (base 8)

Il sistema esadecimale

- base = 16 (talvolta indicata con H per Hexadecimal)
  - cifre = { 0, 1, ..., 9, A, B, C, D, E, F }
  - utile per scrivere in modo compatto i numeri binari (4:1)



Strumenti interattivi



https://integer.exposed/

- `int(stringa)`
  - Converte una stringa in numero intero in base 10
- `int(stringa, base=X)`
  - Converte una stringa in numero intero in base 10
  - `int('344', base=5)` → `99`
- `str(intero)`
  - Converte un numero intero in stringa in base 10
- `bin(intero)`
  - Converte un numero intero in stringa in base 2
  - `bin(17)` → `'0b10001'`
- `oct(intero)`
  - Converte un numero intero in stringa in base 8
- `hex(intero)`
  - Converte un numero intero in stringa in base 16
  - `hex(17)` → `'0x11'`

Python

Rappresentazione dei numeri relativi

I numeri con segno

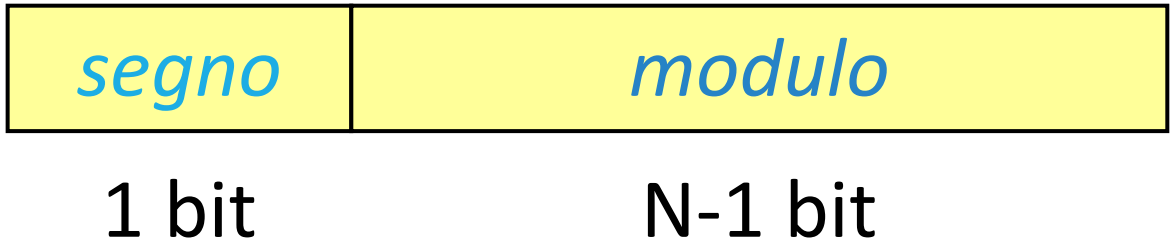
- Il segno dei numeri può essere solo di due tipi:
  - positivo (+)
  - negativo (–)

- È quindi facile rappresentarlo in binario ... ma non sempre la soluzione più semplice è quella migliore!
- Varie soluzioni, le più usate sono
  - Modulo e segno
  - Complemento a due

Codifica “modulo e segno”

- un bit per il segno (tipicamente il MSB):
  - 0 = segno positivo (+)
  - 1 = segno negativo (–)

- N-1 bit per il valore assoluto (anche detto modulo)



Modulo e segno: esempi

- Usando una codifica su quattro bit:

$$+3_{10} \rightarrow 0011_{M\&S}$$
$$-3_{10} \rightarrow 1011_{M\&S}$$
$$0000_{M\&S} \rightarrow +0_{10}$$
$$1000_{M\&S} \rightarrow -0_{10}$$

Modulo e segno

- Svantaggi:
  - doppio zero (+ 0, - 0)
  - operazioni complesse
  - es. somma A+B

	A > 0	A < 0
B > 0	**A + B**	**B - |A|**
B < 0	**A - |B|**	**- (|A| + |B|)**

Modulo e segno: limiti

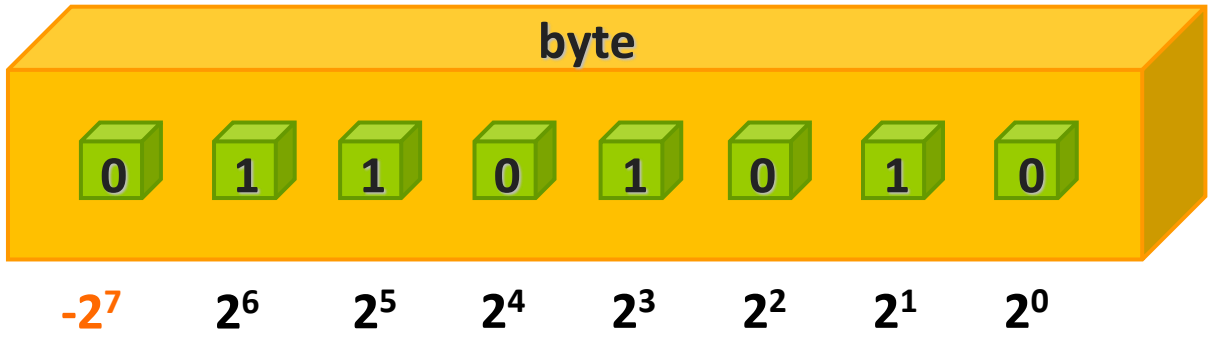
- In una rappresentazione M&S su N bit:

$$-(2^{N-1}-1) \leq x \leq +(2^{N-1}-1)$$

- Esempi:
  - 8 bit = [-127 … +127]
  - 16 bit = [-32 767 … +32 767]

Codifica in complemento a due

- In questa codifica per un numero a N bit:
  - il MSB ha peso negativo (pari a -2^{N-1})
  - gli altri bit hanno peso positivo



- Ne consegue che MSB indica sempre il segno:
  - 0 = + 1 = -

Complemento a due (esempio)

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

-32 +8 +4 +1= -19

I numeri con segno

- **Il segno dei numeri può essere solo di due tipi:**
  - **positivo (+)**
  - **negativo (–)**

- Abbiamo visto due opzioni
  - **Modulo e segno**
    - Semplice da ‘vedere’, poco pratico per le operazioni
  - **Complemento a due**
    - **Pratico per le operazioni**
    - **Usato da (praticamente) tutti i calcolatori**

Da decimale a complemento a 2

- L'idea del complemento a due e' quella di **evitare la netta separazione tra segno e modulo del M&S** e fare in modo che **l'intero valore binario sia posizionale**

- **Chiave: il MSB va considerato con segno negativo !!!**

Complemento a due (esempio)

1	0	1	1	0	1
-2^{55}	2^4	2^3	2^2	2^1	2^0

-32 +8 +4 +1= -19

Dati N bit, il MSB ha peso -2^{N-1}

Da decimale a complemento a 2

- Per convertire un numero decimale in complemento a 2:
- Se **positivo**, si effettua la solita conversione
- Se **negativo**:
  - Si converte il modulo in binario
  - Si complementa ogni bit (0->1, 1->0)
  - Si somma 1 (sul corrispondente numero di bit)

Da decimale a complemento a 2

- Esempio
  - +15 su 5 bit in c.a.2 ⇨ $+15 = 01111_2$ ⇨ **01111**
  - -12 su 5 bit in c.a.2 ⇨ $+12 = 01100_2$
    complementiamo i bit ⇨ 10011
    sommiamo +1 (su 5 bit) ⇨ 10011 +
    00001 =

    10100

Complemento a 2 e operazioni

- La rappresentazione in complemento a due è oggi la più diffusa perché semplifica la realizzazione dei circuiti per eseguire le operazioni aritmetiche
- Possono essere applicate le regole binarie a tutti i bit, segno compreso!

- La somma e sottrazione si effettuano direttamente, senza badare ai segni degli operandi
- La sottrazione si può effettuare sommando al minuendo il CA2 del sottraendo

Somma in CA2 - esempio

00100110 + 11001011

00100110 +
11001011 =

11110001

verifica: 38 + (– 53) = – 15

Sottrazione in CA2 - esempio

00100110 – 11001011

00100110 -
11001011 =

01011011

verifica: 38 – (– 53) = 91

Overflow nella somma in CA2

- Operandi con segno discorde: non si può mai verificare overflow.
- Operandi con segno concorde: c'è overflow quando il risultato ha segno discorde.
- In ogni caso, si trascura sempre il carry sul MSB.

Complemento a 2: limiti

- In una rappresentazione c.a 2 su N bit:

$$-(2^{N-1}) \leq x \leq +(2^{N-1} - 1)$$

- Esempi:
  - 8 bit = [-128 ... +127]
  - 16 bit = [-32 768 ... +32 767]

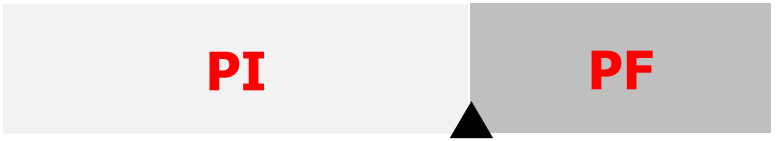
Riepilogo e Limiti della rappresentazione

Codifica	Valore minimo	Valore massimo	-1	0	+1	Overflow
Binario puro	0 000…000	2^N-1 111…111	N/A	000…000	000…001	Carry o Borrow sull'MSB
Modulo e Segno	$-(2^{N-1}-1)$ 111…111	$2^{N-1}-1$ 011…111	100…001	000…000 100…000	000…001	It's complicated…
Complemento a 2	-2^{N-1} 100…000	$2^{N-1}-1$ 011…111	111…111	000…000	000…001	Coerenza del segno del risultato con i segni degli addendi

Rappresentazione di numeri reali

Rappresentazione di numeri reali

- Due opzioni:
  1. Dati N bit disponibili riservarne M per la parte frazionaria e N-M per la parte intera **(VIRGOLA FISSA)**



Virgola “virtuale”

  2. Implementare negli N bit la notazione esponenziale (“scientifica”) **(VIRGOLA MOBILE)**

3.14	$0.314 \times 10^{+1}$
0.000001	0.1×10^{-5}

$$X = \pm M \times 2^{E}$$

Perché virgola mobile?

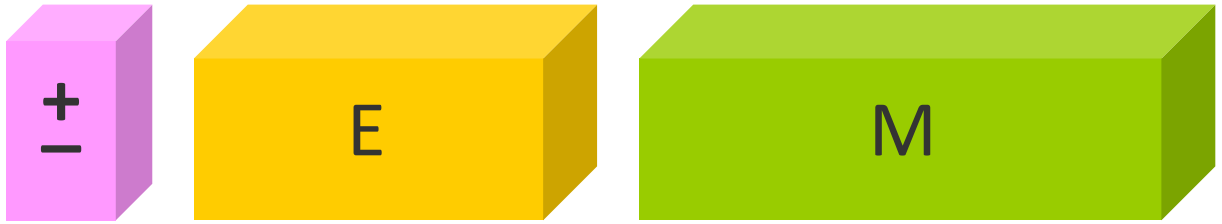
- Virgola **fissa** = si riserva un numero di posizioni (bit) predefinite alla parte intera ed alla parte frazionaria
  - **Precisione fissa**
- NOTA: I bit della parte frazionaria hanno peso $\mathbf{2^{-i}}$
  - E' sempre posizionale!
  - Esempio:
    $(1.23)_{10} = 1*10^{0} + 2*10^{-1} + 3*10^{-2}$
    Allo stesso modo.
    $11.011 = 2^{1} + 2^{0} + 2^{-2} + 2^{-3} = 2 + 1 + 0.25 + 0.125 = 3.375$

- Virgola **mobile** = **precisione variabile**
  - Nella stessa rappresentazione possiamo rappresentare sia numeri molto grandi (esponenti grandi) sia molto piccoli (esponenti piccoli)

Rappresentazione in virgola mobile (Floating Point)

Nella memoria del calcolatore si memorizzano:

- Segno
- Esponente (con il suo segno)
- Mantissa

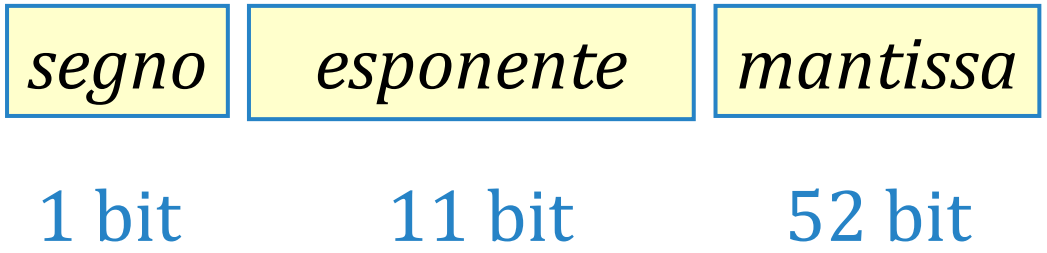


$$X = \pm M \times 2^{E}$$

Formato IEEE-754

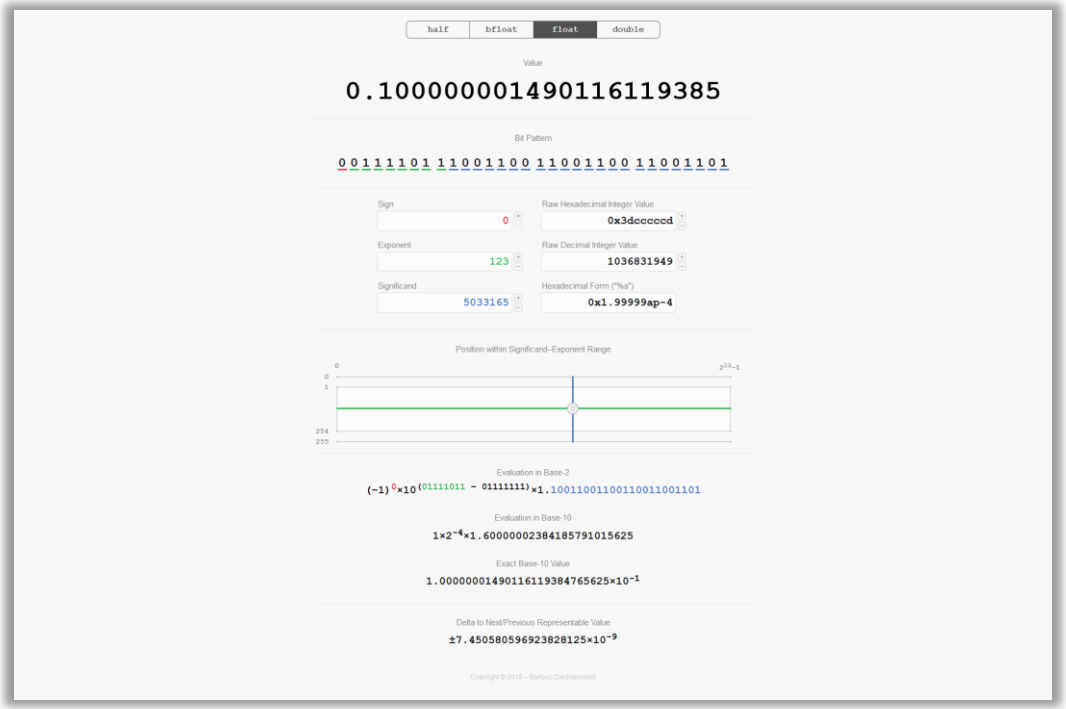
- Mantissa nella forma ‘1,…’ (valore max < 2)
- Base dell’esponente pari a 2
- IEEE 754 SP: (**`float`**)

- IEEE 754 DP: (**`double`**)



Esempi

https://float.exposed/

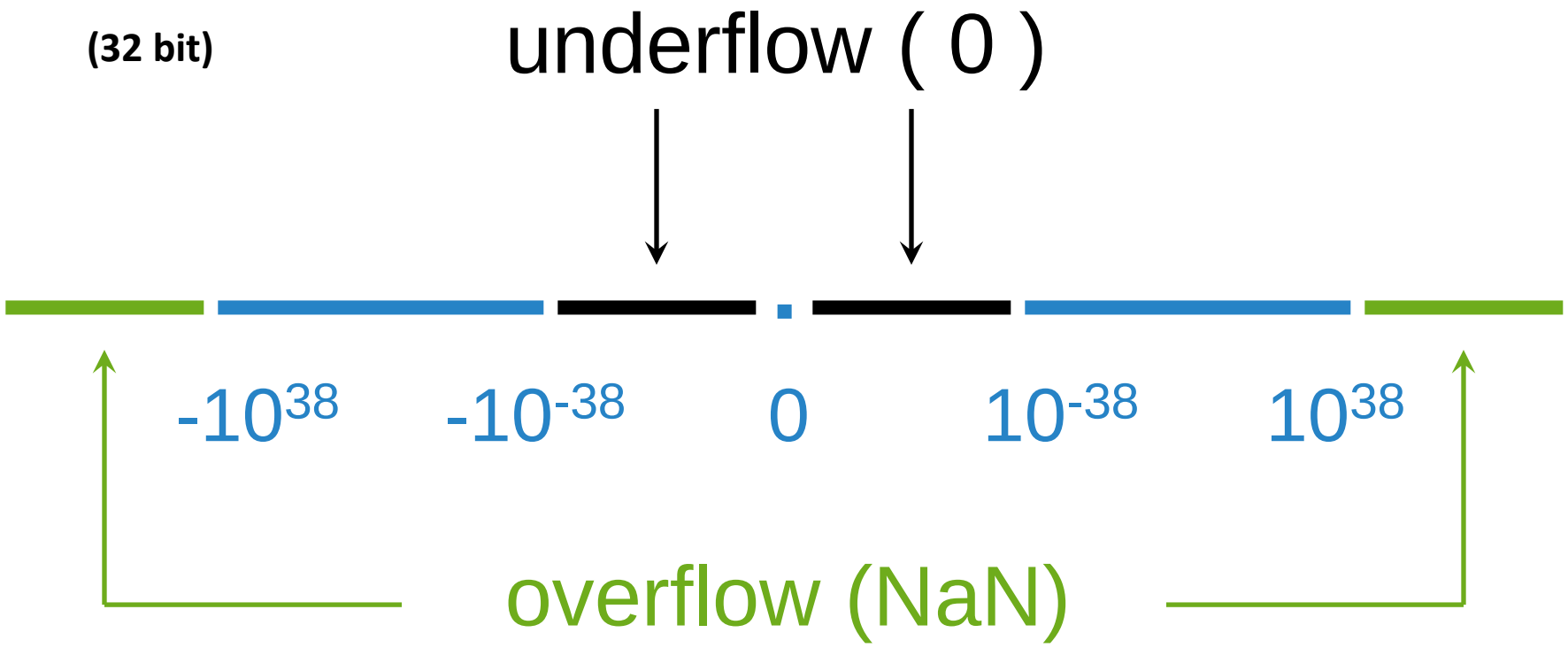


Floating point ed approssimazioni

- La limitatezza della precisione porta ad avere problemi con le operazioni aritmetiche

- **Alcuni numeri NON sono rappresentabili in modo esatto**
  - E non sono numeri ‘strani’…
    - Valori quali 0.1, 0.6 sono **approssimati**

IEEE-754 SP: intervallo di valori



Floating point ed approssimazioni

- La limitatezza della precisione porta ad avere problemi con le operazioni aritmetiche
- Esempio: in FP, **la somma NON e' associativa**!!!
  - x+(y+z) puo' essere diverso da (x+y)+z!

- Esempio:
  - $x = -1.5_{10} * 10^{38}$
  - $y = +1.5_{10} * 10^{38}$
  - $z = 1.0_{10}$
  - Eseguendo su calcolatore
  - $x+(y+z) = -1.5_{10} * 10^{38} + (1.5_{10} * 10^{38} + 1) =$
    $= -1.5_{10} * 10^{38} + 1.5_{10} * 10^{38} = \mathbf{0}$
  - $(x+y)+z = (-1.5_{10} * 10^{38} + 1.5_{10} * 10^{38}) + 1 = \mathbf{1}$

Floating point e linguaggi di programmazione

- Diversamente dai numeri interi, lo standard dei numeri reali IMPONE I limiti di rappresentazione dello standard stesso
  - Tutti i linguaggi principali si adeguano a questo

- In Python, `float` = doppia precisione
  - **VALORI max RAPPRESENTABILI**

    ±1.7976931348623157e308

    - **Fuori da questi range, diventa ±inf**
  - **VALORI min RAPPRESENTABILI**

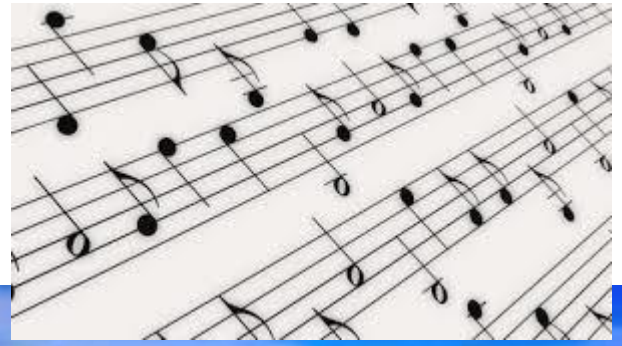
    ± 2.2250738585072014e-308

    - **Fuori da questi range, diventa 0**

Rappresentazione di dati non numerici

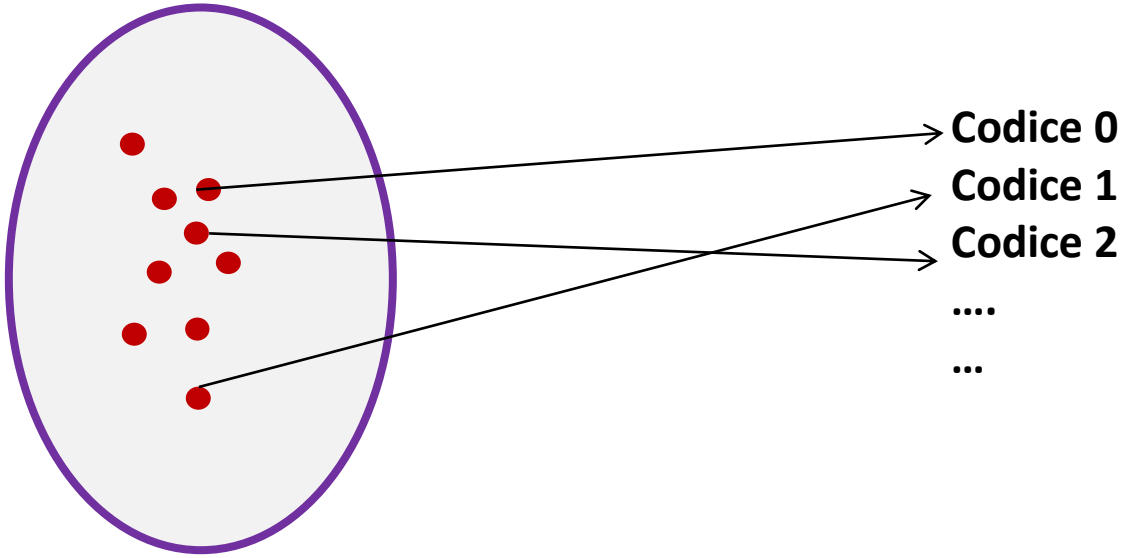
Elaborazione dell'informazione non numerica

Sempre caro mi fu quest'ermo colle,
e questa siepe, che da tanta parte
dell'ultimo orizzonte il guardo esclude.
Ma sedendo e mirando, interminati
spazi di là da quella, e sovrumani
silenzi, e profondissima quiete
io nel pensier mi fingo; ove per poco
il cor non si spaura. E come il vento
odo stormir tra queste piante, io quello
infinito silenzio a questa voce
vo comparando: e mi sovvien l'eterno,
e le morte stagioni, e la presente
e viva, e il suon di lei. Così tra questa
immensità s'annega il pensier mio:
e il naufragar m'è dolce in questo mar.



Informazione non numerica

- Il calcolatore è in grado di manipolare SOLO numeri!
- Per gestire dati non numerici l'unica possibilità è creare una corrispondenza tra oggetti e numeri
  - Ad ogni oggetto si assegna un codice univoco
  - Questo codice diventa la rappresentazione dell'oggetto
    - Nel calcolatore, il codice sara' binario…



Oggetti e numeri

- Assumendo di assegnare codici binari, dati N bit si possono codificare 2N «oggetti» distinti
- Esempio (3 bit):

Codici binari	000	001	010	011	100	101	110	111
oggetti	0	1	2	3	4	5	6	7

- Se viceversa ho M oggetti, per codificarli tutti dovrò usare un numero di bit N pari a $N = \lceil \log_2 M \rceil$
  - In pratica, la prima potenza di 2 tale che $2^N > M$

Codifica dei caratteri: codice ASCII

- Occorre una codifica standard perché è il genere di informazione più scambiata:
  - codice ASCII (American Standard Code for Information Interchange)

- Usa 8 bit (originariamente 7 bit per US-ASCII) per rappresentare:
  - 52 caratteri alfabetici (a...z A...Z)
  - 10 cifre (0...9)
  - segni di interpunzione (,;!?...)
  - caratteri di controllo

Codice 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

Caratteri di controllo

	CR	(13)	Carriage Return
LF,	NL	(10)	New Line, Line Feed
FF,	NP	(12)	New Page, Form Feed
	HT	(9)	Horizontal Tab
	VT	(11)	Vertical Tab
	NUL	(0)	Null
	BEL	(7)	Bell
	EOT	(4)	End-Of-Transmission
	. . .	. . .	. . .

UNICODE e UTF-8

- **Unicode** utilizza 21 bit per carattere ed esprime tutti i caratteri di tutte le lingue del mondo (più di un milione), oltre agli emoji 🤩.
- È il codice usato per rappresentare i caratteri in Python
- **UTF-8** è la codifica di Unicode su file più usata:
  - **1 byte per caratteri US-ASCII (MSB=0)**
  - 2 byte per caratteri Latini con simboli diacritici, Greco, Cirillico, Armeno, Ebraico, Arabo, Siriano e Maldiviano
  - 3 byte per altre lingue di uso comune
  - 4 byte per caratteri rarissima
  - raccomandata da IETF per e-mail

https://home.unicode.org/
https://unicode-table.com/it/

Codifiche o formati di testo/stampa

- Non confondere il formato di un file word, con il codice ASCII!!
- Un testo può essere memorizzato in due formati
  - Formattato: sono memorizzate sequenze di byte che definiscono l'aspetto del testo (e.g., font, spaziatura)
  - Non formattato: sono memorizzati unicamente i caratteri che compongono il testo

Codifiche audio, video, …

- Molto più articolate, ma basate sul solito principio di associazione oggetti <-> codici
  - Per es: I colori sono codificati su 8 bit per canale (R,G,B), quindi fino a 256 sfumature di colore per canale

- Oggetto di corsi più avanzati…