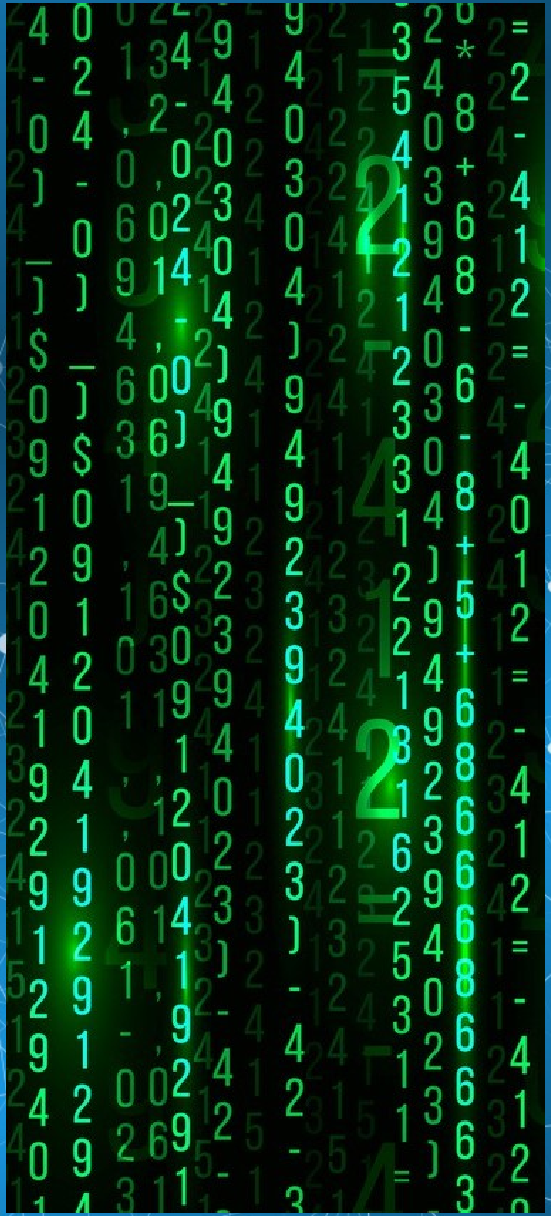
ISLab Intelligent Systems Laboratory

UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

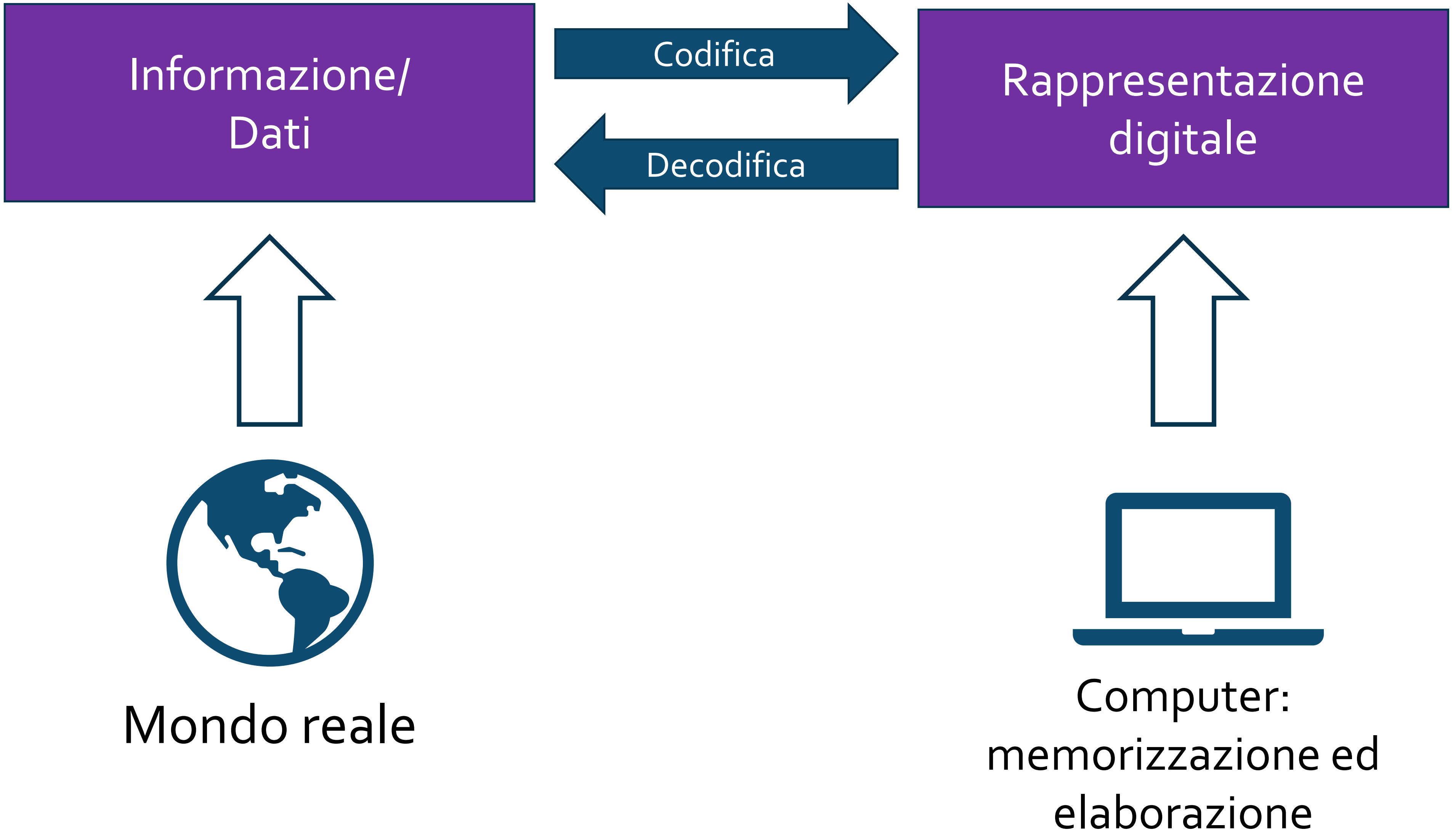
Tipi di dati e loro rappresentazione

Prof. Giuseppe Pirlo | Prof. Donato Impedovo | Dott. Stefano Galantucci

Architettura degli Elaboratori e Sistemi Operativi @ Corso di Laurea in Informatica



Rappresentazione digitale dell’informazione



Il bit

Presenza	Assenza
Vero	Falso
1	0
Acceso	Spento
+	–
Sì	No
Favorevole	Contrario
Yang	Yin
Lisa	Bart

Alfabeto del calcolatore costituito da due simboli: 0,1.

BIT (binary digit):

Unità elementare di informazione. La cifra binaria può assumere solo due valori alternativi: 0 oppure 1. Archiviato da un dispositivo digitale o un sistema fisico che esiste in uno di due possibili stati distinti.

Es.:

- i due stati stabili di un flip-flop
- due posizioni di un interruttore elettrico
- due distinte tensione o gli attuali livelli consentiti da un circuito
- due distinti livelli di intensità della luce
- due direzioni di magnetizzazione o di polarizzazione, ecc

Sequenze di bit

Per poter rappresentare un numero maggiore di informazione si usano sequenze di bit

Il processo che fa corrispondere ad un dato reale una sequenze di bit prende il nome di **codifica dell'informazione**

Es.1: un esame può avere quattro possibili esiti: ottimo, discreto, sufficiente, insufficiente. Quanti bit sono necessari per codificare tale informazione?

Ottimo	0	0
Discreto	0	1
Sufficiente	1	0
Insufficiente	1	1

Es.2: rappresentazione di otto colori

Rosso	0	0	0
Blu	0	0	1
Verde	0	1	0
Giallo	0	1	1
Viola	1	0	0
Bianco	1	0	1
Nero	1	1	0
Grigio	1	1	1

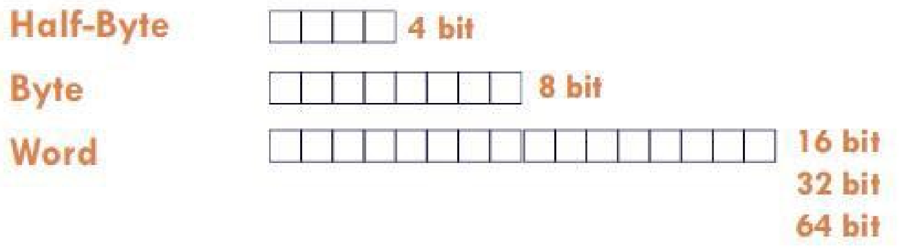
bit, Byte e word

- Con n bit si possono rappresentare 2^n stati/valori differenti
- Per rappresentare n stati/valori, devo usare almeno $\lceil \log_2 n \rceil$

I sistemi moderni memorizzano e manipolano miliardi di bit: vi è quindi la necessità di avere dei multipli

8 bit = 1 byte

Con la lettera b minuscola si indicano i bit, mentre con la lettera B si indicano i byte



Multipli del byte

Byte	1 B =	8 bit	10^0 B
KiloByte	1 KB =	1000 B	10^3 B
MegaByte	1 MB =	1000 KB	10^6 B
GigaByte	1 GB =	1000 MB	10^9 B
TeraByte	1 TB =	1000 GB	10^{12} B
PetaByte	1 PB =	1000 TB	10^{15} B
ExaByte	1 EB =	1000 PB	10^{18} B
ZettaByte	1 ZB =	1000 EB	10^{21} B
YottaByte	1 YB =	1000 ZB	10^{24} B

Byte	1 B =	8 bit	2^0 B
KibiByte	1 KiB =	1024 B	2^{10} B
MebiByte	1 MiB =	1024 KiB	2^{20} B
GibiByte	1 GiB =	1024 MiB	2^{30} B
TebiByte	1 TiB =	1024 GiB	2^{40} B
PebiByte	1 PiB =	1024 TiB	2^{50} B
ExbiByte	1 EiB =	1024 PiB	2^{60} B
ZebiByte	1 ZiB =	1024 EiB	2^{70} B
YobiByte	1 YiB =	1024 ZiB	2^{80} B

Esistono analoghe misure per i multipli dei bit (Kb – KiloBit – 1000 bit), utilizzati generalmente come misura nelle quantità di dati trasmessi

Fino a qualche anno fa le misure sulla sinistra non esistevano, e i nomi di quelle sulla sinistra erano corrispondenti a quelle di destra (cioè 1 KiloByte equivaleva a 1024 Byte)

Tipi di dati non numerici

Booleani: i dati booleani sono contenuti all’interno di singoli bit che assumono valore 0/1 – F/V. Comunemente si considera come falso il valore 0 e come vero qualsiasi altro valore.

Ad esempio:
if(a) corrisponde a if(a!=0)

In questa situazione il valore viene memorizzato all’interno di uno (ad esempio per il *char*) o più byte (*short, int…*).

L’unico caso in cui un booleano viene effettivamente memorizzato in un singolo bit si ha quando si utilizzano le **mappe di bit**: viene considerato un byte come se fosse un array di bit, ciascuno dei quali rappresenta un valore booleano.

Tipi di dati non numerici

Caratteri: Mappati come interi equivalenti in ASCII/UNICODE

In particolare si ha

UNICODE:

UTF-8	8 bit (1 byte)	ASCII esteso
UTF-16	16 bit (2 byte)	Espansione a linguaggi occidentali
UTF-32	32 bit (4 byte)	Set più completo di caratteri

ASCII invece occupa 7 bit

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

Full Unicode Table

ᎌ	ᎌ
֎	֎
ᅋ	ᅋ

ISLab

Click on the entity to add it to the list. Then you can copy it by clicking it in the list.

Press F3 or CTRL+F to find an entity.

30 columns ▾

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	�	□	□	□	□	□	□	□	□						□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
30	□	□		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{			}	~		€	□	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	□	Ž	□	□	‘	’	“	”	•
150	–	—	˜	™	š	›	œ	□	ž	Ÿ		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯	°	±	²	³
180	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	Ā	ā	Ă	ă	Ą	ą	Ć	ć	Ĉ	ĉ	Ċ	ċ	Č	č
270	Ď	ď	Đ	đ	Ē	ē	Ĕ	ĕ	Ė	ė	Ę	ę	Ě	ě	Ĝ	ĝ	Ğ	ğ	Ġ	ġ	Ģ	ģ	Ĥ	ĥ	Ħ	ħ	Ĩ	ĩ	Ī	ī
300	Ĭ	ĭ	Į	į	İ	ı	Ĳ	ĳ	Ĵ	ĵ	Ķ	ķ	ĸ	Ĺ	ĺ	Ļ	ļ	Ľ	ľ	Ŀ	ŀ	Ł	ł	Ń	ń	Ņ	ņ	Ň	ň	ŉ
330	Ŋ	ŋ	Ō	ō	Ŏ	ŏ	Ő	ő	Œ	œ	Ŕ	ŕ	Ŗ	ŗ	Ř	ř	Ś	ś	Ŝ	ŝ	Ş	ş	Š	š	Ţ	ţ	Ť	ť	Ŧ	ŧ
360	Ũ	ũ	Ū	ū	Ŭ	ŭ	Ů	ů	Ű	ű	Ų	ų	Ŵ	ŵ	Ŷ	ŷ	Ÿ	Ź	ź	Ż	ż	Ž	ž	ſ	ƀ	Ɓ	Ƃ	ƃ	Ƅ	ƅ
390	Ɔ	Ƈ	ƈ	Ɖ	Ɗ	Ƌ	ƌ	ƍ	Ǝ	Ə	Ɛ	Ƒ	ƒ	Ɠ	Ɣ	ƕ	Ɩ	Ɨ	Ƙ	ƙ	ƚ	ƛ	Ɯ	Ɲ	ƞ	Ɵ	Ơ	ơ	Ƣ	ƣ
420	Ƥ	ƥ	Ʀ	Ƨ	ƨ	Ʃ	ƪ	ƫ	Ƭ	ƭ	Ʈ	Ư	ư	Ʊ	Ʋ	Ƴ	ƴ	Ƶ	ƶ	Ʒ	Ƹ	ƹ	ƺ	ƻ	Ƽ	ƽ	ƾ	ƿ	ǀ	ǁ
450	ǂ	ǃ	Ǆ	ǅ	ǆ	Ǉ	ǈ	ǉ	Ǌ	ǋ	ǌ	Ǎ	ǎ	Ǐ	ǐ	Ǒ	ǒ	Ǔ	ǔ	Ǖ	ǖ	Ǘ	ǘ	Ǚ	ǚ	Ǜ	ǜ	ǝ	Ǟ	ǟ
480	Ǡ	ǡ	Ǣ	ǣ	Ǥ	ǥ	Ǧ	ǧ	Ǩ	ǩ	Ǫ	ǫ	Ǭ	ǭ	Ǯ	ǯ	ǰ	Ǳ	ǲ	ǳ	Ǵ	ǵ	Ƕ	Ƿ	Ǹ	ǹ	Ǻ	ǻ	Ǽ	ǽ
510	Ǿ	ǿ	Ȁ	ȁ	Ȃ	ȃ	Ȅ	ȅ	Ȇ	ȇ	Ȉ	ȉ	Ȋ	ȋ	Ȍ	ȍ	Ȏ	ȏ	Ȑ	ȑ	Ȓ	ȓ	Ȕ	ȕ	Ȗ	ȗ	Ș	ș	Ț	ț
540	Ȝ	ȝ	Ȟ	ȟ	Ƞ	ȡ	Ȣ	ȣ	Ȥ	ȥ	Ȧ	ȧ	Ȩ	ȩ	Ȫ	ȫ	Ȭ	ȭ	Ȯ	ȯ	Ȱ	ȱ	Ȳ	ȳ	ȴ	ȵ	ȶ	ȷ	ȸ	ȹ
570	Ⱥ	Ȼ	ȼ	Ƚ	Ⱦ	ȿ	ɀ	Ɂ	ɂ	Ƀ	Ʉ	Ʌ	Ɇ	ɇ	Ɉ	ɉ	Ɋ	ɋ	Ɍ	ɍ	Ɏ	ɏ	ɐ	ɑ	ɒ	ɓ	ɔ	ɕ	ɖ	ɗ
600	ɘ	ə	ɚ	ɛ	ɜ	ɝ	ɞ	ɟ	ɠ	ɡ	ɢ	ɣ	ɤ	ɥ	ɦ	ɧ	ɨ	ɩ	ɪ	ɫ	ɬ	ɭ	ɮ	ɯ	ɰ	ɱ	ɲ	ɳ	ɴ	ɵ
630	ɶ	ɷ	ɸ	ɹ	ɺ	ɻ	ɼ	ɽ	ɾ	ɿ	ʀ	ʁ	ʂ	ʃ	ʄ	ʅ	ʆ	ʇ	ʈ	ʉ	ʊ	ʋ	ʌ	ʍ	ʎ	ʏ	ʐ	ʑ	ʒ	ʓ
660	ʔ	ʕ	ʖ	ʗ	ʘ	ʙ	ʚ	ʛ	ʜ	ʝ	ʞ	ʟ	ʠ	ʡ	ʢ	ʣ	ʤ	ʥ	ʦ	ʧ	ʨ	ʩ	ʪ	ʫ	ʬ	ʭ	ʮ	ʯ	ʰ	ʱ
690	ʲ	ʳ	ʴ	ʵ	ʶ	ʷ	ʸ	ʹ	ʺ	ʻ	ʼ	ʽ	ʾ	ʿ	ˀ	ˁ	˂	˃	˄	˅	ˆ	ˇ	ˈ	ˉ	ˊ	ˋ	ˌ	ˍ	ˎ	ˏ
720	ː	ˑ	˒	˓	˔	˕	˖	˗	˘	˙	˚	˛	˜	˝	˞	˟	ˠ	ˡ	ˢ	ˣ	ˤ	˥	˦	˧	˨	˩	˪	˫	ˬ	˭
750	ˮ	˯	˰	˱	˲	˳	˴	˵	˶	˷	˸	˹	˺	˻	˼	˽	˾	˿	̀	́	̂	̃	̄	̅	̆	̇	̈	̉	̊	̋
780	̌	̍	̎	̏	̐	̑	̒	̓	̔	̕	̖	̗	̘	̙	̚	̛	̜	̝	̞	̟	̠	̡	̢	̣	̤	̥	̦	̧	̨	̩
810	̪	̫	̬	̭	̮	̯	̰	̱	̲	̳	̴	̵	̶	̷	̸	̹	̺	̻	̼	̽	̾	̿	̀	́	͂	̓	̈́	ͅ	͆	͇
840	͈	͉	͊	͋	͌	͍	͎	͏	͐	͑	͒	͓	͔	͕	͖	͗	͘	͙	͚	͛	͜	͝	͞	͟	͠	͡	͢	ͣ	ͤ	ͥ
870	ͦ	ͧ	ͨ	ͩ	ͪ	ͫ	ͬ	ͭ	ͮ	ͯ	Ͱ	ͱ	Ͳ	ͳ	ʹ	͵	Ͷ	ͷ	□	□	ͺ	ͻ	ͼ	ͽ	;	Ϳ	□	□	□	□
900	΄	΅	Ά	·	Έ	Ή	Ί	□	Ό	□	Ύ	Ώ	ΐ	Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο	Π	Ρ
930	□	Σ	Τ	Υ	Φ	Χ	Ψ	Ω	Ϊ	Ϋ	ά	έ	ή	ί	ΰ	α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο
960	π	ρ	ς	σ	τ	υ	φ	χ	ψ	ω	ϊ	ϋ	ό	ύ	ώ	Ϗ	ϐ	ϑ	ϒ	ϓ	ϔ	ϕ	ϖ	ϗ	Ϙ	ϙ	Ϛ	ϛ	Ϝ	ϝ
				ϡ	Ϣ	ϣ	Ϥ	ϥ	Ϧ	ϧ	Ϩ	ϩ	Ϫ	ϫ	Ϭ	ϭ	Ϯ	ϯ	ϰ	ϱ	ϲ	ϳ	ϴ	ϵ	϶	Ϸ	ϸ	Ϲ	Ϻ	ϻ
				Ͽ	Ѐ	Ё	Ђ	Ѓ	Є	Ѕ	І	Ї	Ј	Љ	Њ	Ћ	Ќ	Ѝ	Ў	Џ	А	Б	В	Г	Д	Е	Ж	З	И	Й
				Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	а	б	в	г	д	е	ж	з
					м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	ѐ	ё	ђ	ѓ	є	ѕ

Select or write name of Font-Family:

Times New Roman ▾

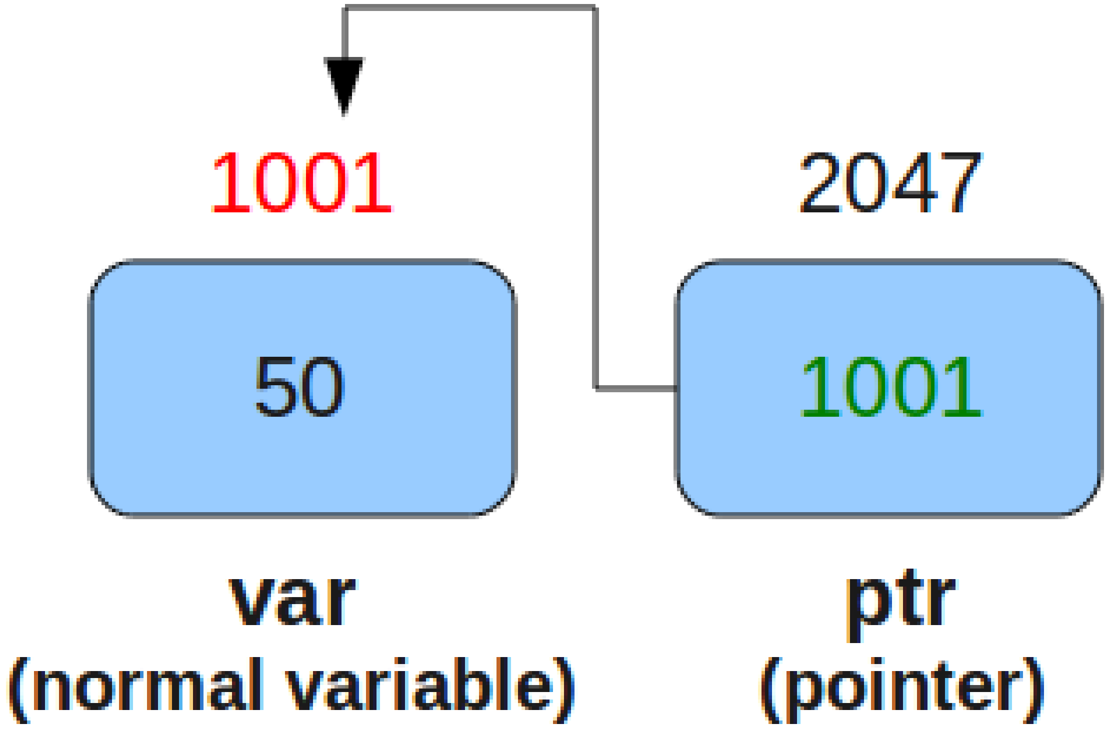
Font Family

☐ Bold ◉ Normal ○ Italic ○ Oblique

Tipi di dati non numerici

Puntatori: rappresentano e memorizzano delle locazioni in memoria.

Lo spazio occupato per un puntatore dipende dalla dimensione dello spazio di indirizzamento.



Tipi di dati numerici

La rappresentazione dei numeri, così come tutte le altre rappresentazioni dei dati, in informatica, a livello circuitale, avviene per tramite del **codice binario**.

Le unità in memoria sono valori binari (corrispondenti ai bit)

Basi numeriche

ISLab Intelligent Systems Laboratory

Base numerica

Il nostro sistema numerico è in **base 10**

Un numero n si denota come scritto in una certa base numerica b mediante la seguente notazione:

$$n_b$$

Ad esempio:

15_{10} indica il numero 15 in base 10
010001_2 indica il numero 010001 in base 2
$23C_{16}$ indica il numero $23C$ in base 16

Base numerica

Un numero scritto in base b può essere composto unicamente dalle cifre comprese tra 0 e $b - 1$ incluse

Per le basi superiori a 10 si usano le lettere dell'alfabeto per indicare le cifre successive. Ad esempio, in base 16 si usano le seguenti cifre:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Le basi più utilizzate in informatica (oltre alla base 10) sono:

- La base 2, detto **sistema binario**
  Le stringhe binarie sono generalmente indicate con il prefisso 0b (ad esempio 0b**01001010**)
- La base 8, detto **sistema ottale**
- La base 16, detto **sistema esadecimale** (HEX)
  Le stringhe esadecimali sono generalmente indicate con il prefisso 0x (ad esempio 0x**FF2C9A**)

Conversione da base b in base 10

Un numero si converte in base 10 mediante la seguente formula

Sia $k_b = (c_{N-1} \dots c_3 c_2 c_1 c_0)_b$ un numero di n cifre in base b, dove c_i rappresenta l'i-esima cifra in base b partendo da destra

$$k_{10} = \sum_{i=0}^{n-1} c_i b^i = (c_0 \cdot b^0) + (c_1 \cdot b^1) + \cdots + (c_{n-1} \cdot b^{n-1})$$

Conversione da base b in base 10

Esempio:

$$3FC2_{16} = ?_{10}$$

$$3 \cdot 16^3 + F \cdot 16^2 + C \cdot 16^1 + 2 \cdot 16^0$$

$$3 \cdot 16^3 + 15 \cdot 16^2 + 12 \cdot 16^1 + 2 \cdot 16^0$$

$$3 \cdot 4096 + 15 \cdot 256 + 12 \cdot 16 + 2 \cdot 1$$

$$= \mathbf{16322_{10}}$$

Conversione da base b in base 10

Esempio:

$$10010101_2 = ?_{10}$$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	**0**	**0**	**1**	**0**	**1**	**0**	**1**
128	-	-	16	-	4	-	1

$$128 + 16 + 4 + 1 = \mathbf{149_{10}}$$

Conversione da base 10 a base b

Si prende il numero e lo si divide per la base b in forma di quoto e resto, successivamente si divide il quoto come prima e si continua fin quando non si ottiene il valore 0 come quoto.
Il numero in base b è rappresentato dai singoli resti, presi come cifre, considerando l'ultimo resto come cifra più significativa e il primo resto come cifra meno significativa.

ALGORITMO:

Operazione modulo (resto nella divisione)

- $i = 0$
- Finché $n \neq 0$:
    - $c_i = n \,\%\, b$
    - $n = \lfloor n/b \rfloor$
    - $i = i + 1$

Conversione da base 10 a base b

Esempio:

$$3425_{10} = ?_{16}$$

	Valore Dividendo	Operazione	Quoto	Resto
1.	3425	3425/16	214	**1**
2.	214	214/16	13	**6**
3.	13	13/16	0	**D** (13)

FINE ALGORITMO

$$\boldsymbol{D61_{16}}$$

Conversione da base 10 a base b

Esempio:

$$213_{10} = ?_2$$

213	**1**
106	**0**
53	**1**
26	**0**
13	**1**
6	**0**
3	**1**
1	**1**
0	

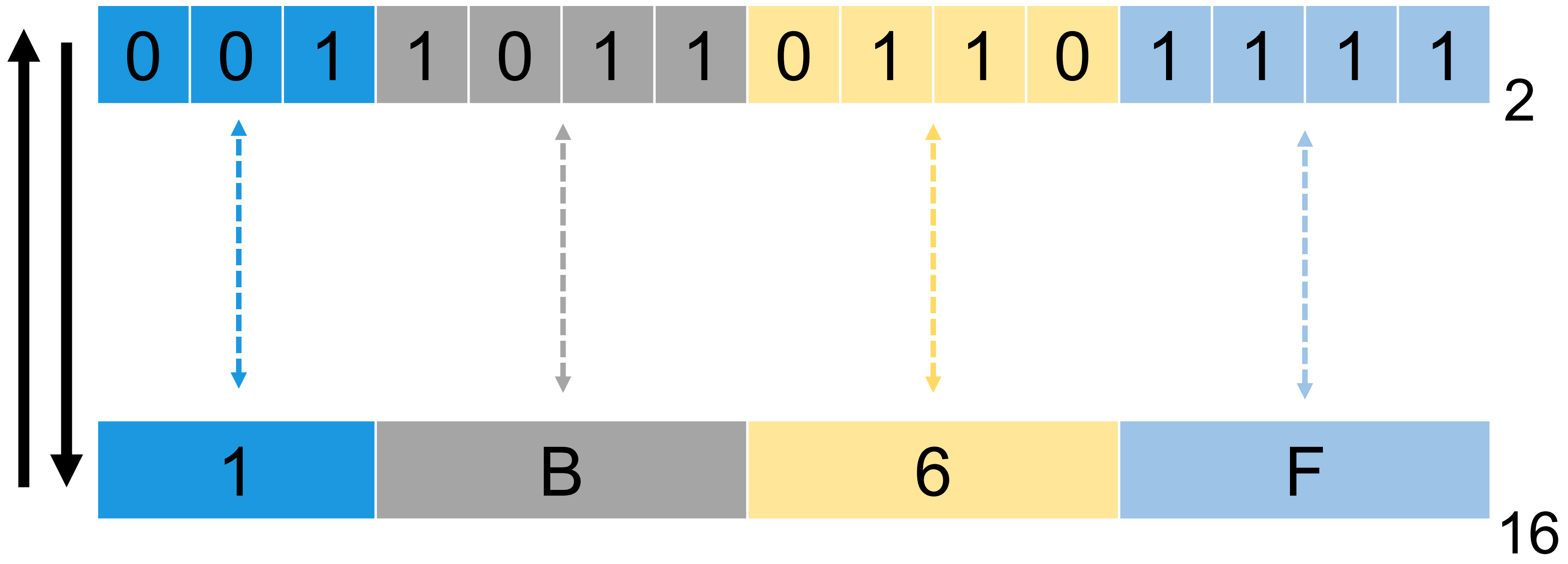
$\mathbf{11010101_2}$

Dividendo per 2 è immediato ricavare il resto:

0 se il numero è pari
1 se il numero è dispari

Conversione rapida da base 2^n a base 2 e viceversa

Essendo da/verso la base $16 = 2^4$ divido in blocchi da 4 cifre a partire da destra e trasformo il blocco da una base all'altra



Rappresentazione di un numero

Per rappresentare un numero $n \in \mathbb{N}$ in base b sono necessarie $\lceil \log_b n \rceil$ cifre

Quindi, per rappresentare un numero $n \in \mathbb{N}$ in binario sono necessari $\lceil \log_2 n \rceil$ bit

Operazioni di somma e sottrazione

Nel sistema binario le operazioni di somma e sottrazione si effettuano nella stessa maniera in cui le effettueremmo in base 10, considerando però i riporti alla base 2:

$$
\begin{array}{rr}
 & 11 \\
22_{10} & 10110+ \\
7_{10} & 00111= \\
\hline
29_{10} & 11101
\end{array}
\qquad
\begin{array}{rr}
 & 0100 \\
22_{10} & \cancel{1}\cancel{0}\cancel{1}\cancel{1}0- \\
7_{10} & 00111= \\
\hline
15_{10} & 01111
\end{array}
$$

Rappresentazione dei numeri relativi

Rappresentazione in modulo e segno

Modulo e segno: viene destinato il bit più significativo al segno (0 = +, 1 = -) e i restanti bit al modulo del numero

Esempio:

segno	modulo						
1	1	0	1	0	1	0	1

$$-85$$

Con la rappresentazione modulo e segno in n bit si possono rappresentare $2^n - 1$ numeri così divisi:

- 2^{n-1} numeri positivi (incluso lo zero)
- 2^{n-1} numeri negativi (incluso lo zero)

L'intervallo rappresentabile è $[-2^{n-1}+1,\ 2^{n-1}-1]_{10}$

Rappresentazione in modulo e segno

Se ho a disposizione n bit posso normalmente rappresentare 2^n valori, ma con il modulo e segno ne riesco a rappresentare $2^n - 1$

Ciò è dovuto alla doppia rappresentazione del valore zero:

segno	modulo						
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Che corrispondono rispettivamente a $+0$ e -0

Operazioni di somma e sottrazione

- Vedremo di seguito come vengono effettuate le operazioni di addizione e sottrazione nella nostra rappresentazione dei dati
- Le operazioni di sottrazione possono essere sempre ottenute invertendo il sottraendo:

$$A - B = A + (-B)$$

- Vi saranno casi in cui le operazioni produrranno un overflow in termini di bit. Laddove i segni siano concordi è necessario valutare se la somma dei moduli (sia essa positiva o negativa) costituisca un valore rappresentabile con il numero di bit scelti per la rappresentazione:
  - Se il modulo è troppo grande per essere rappresentato si ha una situazione di overflow (se gli operandi sono positivi) o di underflow (se gli operandi sono negativi) – le operazioni produrranno dunque un <u>risultato errato</u>
  - Se invece il modulo può essere rappresentato, il bit in eccesso dovrà essere gestito correttamente per garantire la validità del risultato
- Il bit in eccesso si può presentare anche nei casi in cui i segni sono discordi, in tal caso dovrà essere gestito correttamente per garantire la validità del risultato

Rappresentazione in modulo e segno

Somma e sottrazione in modulo e segno:
Per effettuare le operazioni di somma e sottrazione tra numeri in modulo e segno è necessario rimuovere il bit di segno e procedere come segue sulla base del segno:

Ipotizzando di voler effettuare la somma $A + B$

	$A > 0$	$A < 0$
$B > 0$	$\|A\| + \|B\|$	Se $\|B\| < \|A\|$ allora $-(\|A\| - \|B\|)$ Se $\|B\| > \|A\|$ allora $\|B\| - \|A\|$
$B < 0$	Se $\|A\| < \|B\|$ allora $-(\|B\| - \|A\|)$ Se $\|A\| > \|B\|$ allora $\|A\| - \|B\|$	$-(\|A\| + \|B\|)$

Svantaggio: eccessivamente complicato

Rappresentazione in complemento a uno

Complemento a uno: i numeri negativi vengono rappresentati tramite il complemento della loro rappresentazione positiva

Esempio:

decimale	binario							
+89	0 ↓	1 ↓	0 ↓	1 ↓	1 ↓	0 ↓	0 ↓	1 ↓
-89	1	0	1	0	0	1	1	0

Posso rappresentare anche qui $2^n - 1$ valori, in quanto vi è la doppia rappresentazione dello 0: 00000000 e 11111111

L'intervallo rappresentabile è anche qui $[-2^{n-1} + 1,\ 2^{n-1} - 1]_{10}$

<u>Il primo bit non viene comunque utilizzato per rappresentare il modulo (in quanto se fosse 1 supererebbe il valore massimo rappresentabile), quindi può essere utilizzato per identificare il segno</u>

Rappresentazione in complemento a uno

Somma e sottrazione in complemento a uno:
Si procede sommando normalmente i valori, e, laddove l'operazione produca un riporto successivo al bit del segno, quest'ultimo viene aggiunto al risultato:

decimale	binario						
riporti	1	1		1			
+10		0	1	0	1	0	+
-4		1	1	0	1	1	=
	1	0	0	1	0	1	
		0	0	1	0	1	+
						1	=
+6		0	0	1	1	0	

Riporto in eccesso (overflow di bit)

Svantaggio: il riporto in eccesso va sommato, quindi potrei avere una somma in più da effettuare

Vantaggio: evidentemente più comodo del modulo e segno

Rappresentazione in complemento a due

Complemento a due: i numeri negativi vengono rappresentati con il complemento a uno incrementato di 1

Esempio:

decimale	binario							
+90	0	1	0	1	1	0	1	0
-90 (Ca1)	1	0	1	0	0	1	0	1
-90 (Ca2)	1	0	1	0	0	1	1	0

<u>Il primo bit non viene comunque utilizzato per rappresentare il modulo (in quanto se fosse 1 supererebbe il valore massimo rappresentabile), quindi può essere utilizzato per identificare il segno</u>

Confronto tra rappresentazioni su 4 bit

<table>
<tr><th>Decimale</th><th>Senza segno</th><th>Modulo e segno</th><th>Complemento a uno</th><th>Complemento a due</th></tr>
<tr><td>8</td><td>1000</td><td>n/d</td><td>n/d</td><td>n/d</td></tr>
<tr><td>7</td><td>111</td><td>0111</td><td>0111</td><td>0111</td></tr>
<tr><td>6</td><td>110</td><td>0110</td><td>0110</td><td>0110</td></tr>
<tr><td>5</td><td>101</td><td>0101</td><td>0101</td><td>0101</td></tr>
<tr><td>4</td><td>100</td><td>0100</td><td>0100</td><td>0100</td></tr>
<tr><td>3</td><td>11</td><td>0011</td><td>0011</td><td>0011</td></tr>
<tr><td>2</td><td>10</td><td>0010</td><td>0010</td><td>0010</td></tr>
<tr><td>1</td><td>1</td><td>0001</td><td>0001</td><td>0001</td></tr>
<tr><td>(+)0</td><td>0</td><td>0000</td><td>0000</td><td rowspan="2">0000</td></tr>
<tr><td>(−)0</td><td>n/d</td><td>1000</td><td>1111</td></tr>
<tr><td>−1</td><td>n/d</td><td>1001</td><td>1110</td><td>1111</td></tr>
<tr><td>−2</td><td>n/d</td><td>1010</td><td>1101</td><td>1110</td></tr>
<tr><td>−3</td><td>n/d</td><td>1011</td><td>1100</td><td>1101</td></tr>
<tr><td>−4</td><td>n/d</td><td>1100</td><td>1011</td><td>1100</td></tr>
<tr><td>−5</td><td>n/d</td><td>1101</td><td>1010</td><td>1011</td></tr>
<tr><td>−6</td><td>n/d</td><td>1110</td><td>1001</td><td>1010</td></tr>
<tr><td>−7</td><td>n/d</td><td>1111</td><td>1000</td><td>1001</td></tr>
<tr><td>−8</td><td>n/d</td><td>n/d</td><td>n/d</td><td>1000</td></tr>
</table>

Vantaggi del complemento a due

Vantaggio n° 1a: unica rappresentazione dello zero

decimale	binario								
0		0	0	0	0	0	0	0	0
-0 (Ca1)		1	1	1	1	1	1	1	1
-0 (Ca2)	~~1~~	0	0	0	0	0	0	0	0

Si può notare che la rappresentazione di $+0$ e -0 si equivalgono, in quanto la somma finale genera un overflow che non viene considerato in quanto sfora il numero di bit in considerazione

Vantaggi del complemento a due

Vantaggio n° 1b: uso al massimo delle rappresentazioni possibili

Questo vantaggio deriva dal fatto che, contestualmente al fatto che lo zero viene rappresentato una sola volta, la rappresentazione «aggiuntiva» occupata dallo zero nel Ca1 diventa la rappresentazione di un altro valore

In particolare:

- lo zero resta corrispondente solo alla notazione composta da tutti 0
- la notazione composta da tutti 1 corrisponde invece al valore -1
- Viene rappresentato un valore negativo in più: -2^{n-1}

In complemento a due su n bit sono quindi rappresentati 2^n valori

L'intervallo rappresentabile è quindi $[-2^{n-1},\ 2^{n-1}-1]_{10}$

Rappresentazione in complemento a due su 4 bit

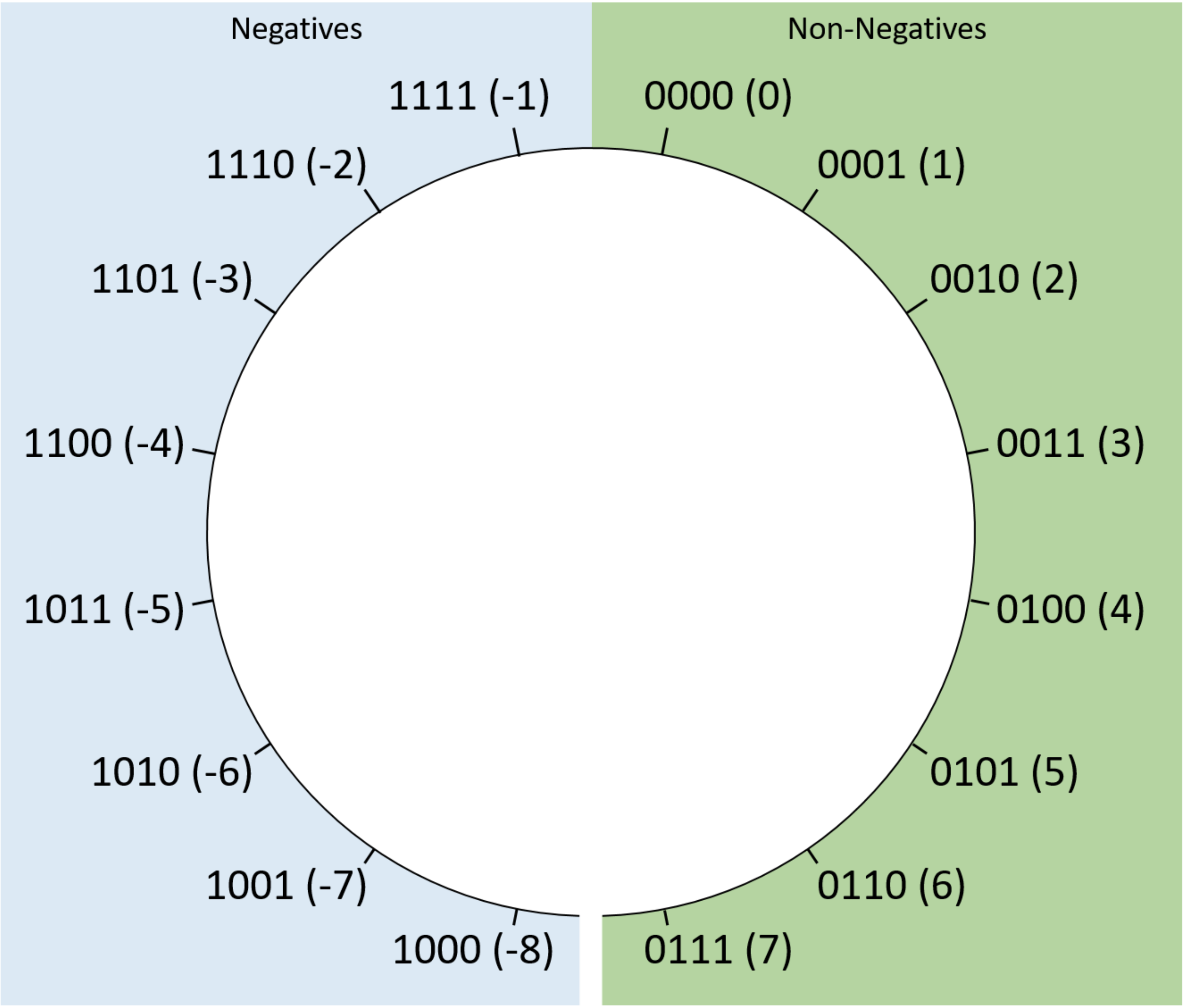
decimale	binario			
-8	1	0	0	0
-7	1	0	0	1
-6	1	0	1	0
-5	1	0	1	1
-4	1	1	0	0
-3	1	1	0	1
-2	1	1	1	0
-1	1	1	1	1
0	0	0	0	0
+1	0	0	0	1
+2	0	0	1	0
+3	0	0	1	1
+4	0	1	0	0
+5	0	1	0	1
+6	0	1	1	0
+7	0	1	1	1

in ordine di valore

decimale	binario			
+0	0	0	0	0
+1	0	0	0	1
+2	0	0	1	0
+3	0	0	1	1
+4	0	1	0	0
+5	0	1	0	1
+6	0	1	1	0
+7	0	1	1	1
-8	1	0	0	0
-7	1	0	0	1
-6	1	0	1	0
-5	1	0	1	1
-4	1	1	0	0
-3	1	1	0	1
-2	1	1	1	0
-1	1	1	1	1

in ordine di rappresentazione binaria

Rappresentazione in complemento a due su 4 bit



Rappresentazione in complemento a due

Il valore decimale di un numero negativo in complemento a due può essere ricavato velocemente tramite il seguente trucco

Considero solo i valori 1 senza considerare il bit di segno.
Sommo al valore di potenza di 2 negato corrispondente al bit più significativo (cioè -2^{n-1} su Ca2 a n bit) i valori di potenza di due corrispondenti ai bit avvalorati a 1:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	**0**	**1**	**0**	**1**	**1**	**0**	**1**
-128		+32		+8	+4		+1

$$-128 + 32 + 8 + 4 + 1 = \mathbf{-83}$$

Il valore di un numero binario su n bit composto dai bit $c_{N-1}c_{N-2} \dots c_1c_0$ in complemento a due è equivalente infatti a:

$$-2^{n-1}c_{n-1} + \sum_{i=0}^{n-2} c_i 2^i$$

Rappresentazione in complemento a due

Somma e sottrazione in complemento a due: si procede normalmente ignorando l'eventuale bit di overflow

decimale	binario						
riporti	1	1		1			
+10		0	1	0	1	0	+
-6		1	1	0	1	0	=
	~~1~~	0	0	1	0	0	
+4		0	0	1	0	0	

Vantaggio n° 2 del complemento a due: somme e sottrazioni possono essere ottenute senza particolari problemi

Bug dell’anno 2038

Binary : 01111111 11111111 11111111 11110000

Decimal : 2147483632

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

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

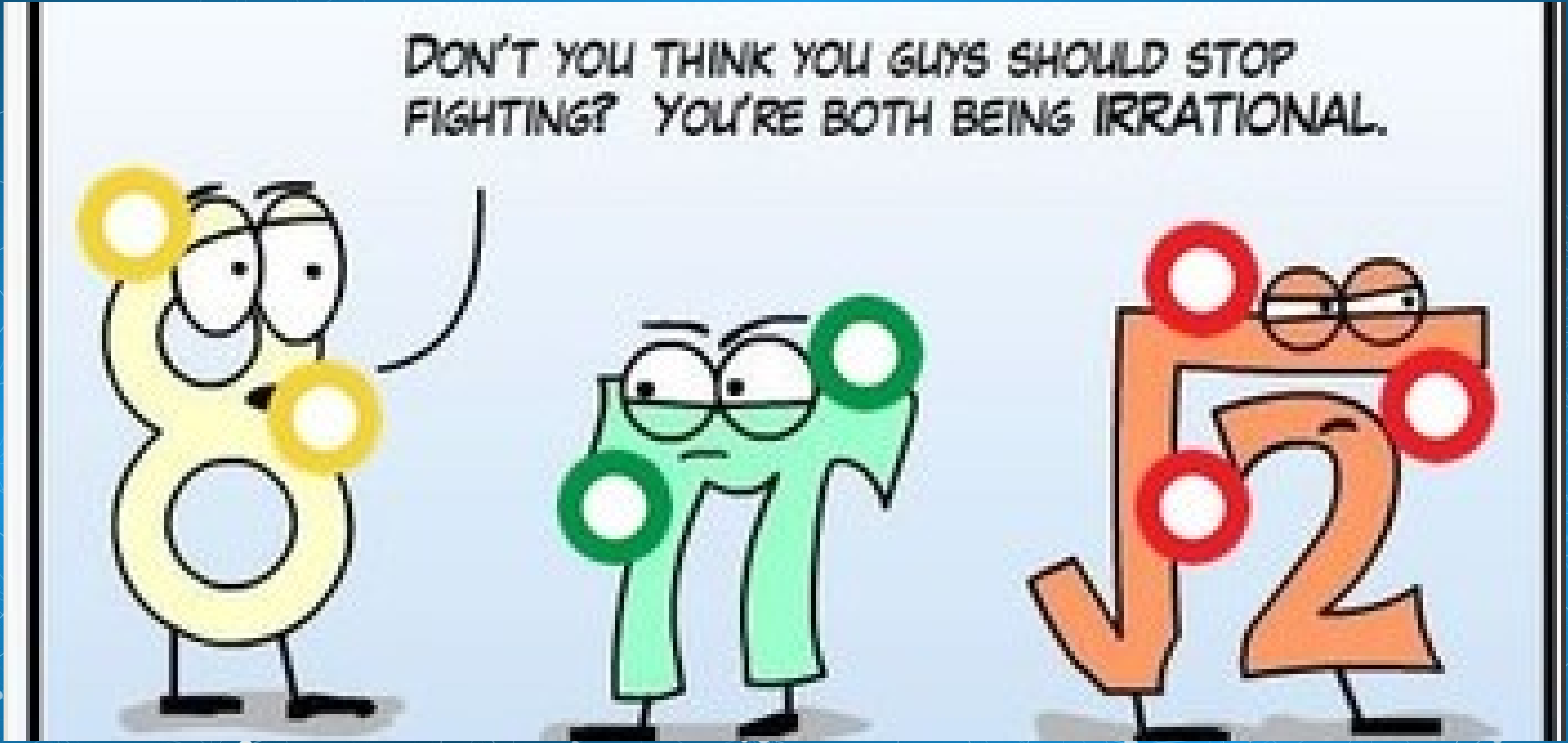
Nel complemento a due, come abbiamo potuto notare precedentemente, sommando 1 al numero più grande rappresentabile si ottiene il numero più piccolo rappresentabile

Questo effetto indesiderato può procurare bug nei sistemi che gestiscono il tempo come numero intero:

- Ad esempio, nei sistemi UNIX, il tempo è considerato come il numero di secondi a partire dal capodanno del 1970 (rappresentazione POSIX)

Poiché tale valore di tempo veniva memorizzato in variabili da 32 bit, giunti al massimo valore rappresentabile, il secondo immediatamente successivo viene interpretato come il minimo valore rappresentabile su 32 bit:

https://it.wikipedia.org/wiki/Bug_dell%27anno_2038



Rappresentazione dei numeri razionali

ISLab Intelligent Systems Laboratory

Rappresentazione dei numeri razionali

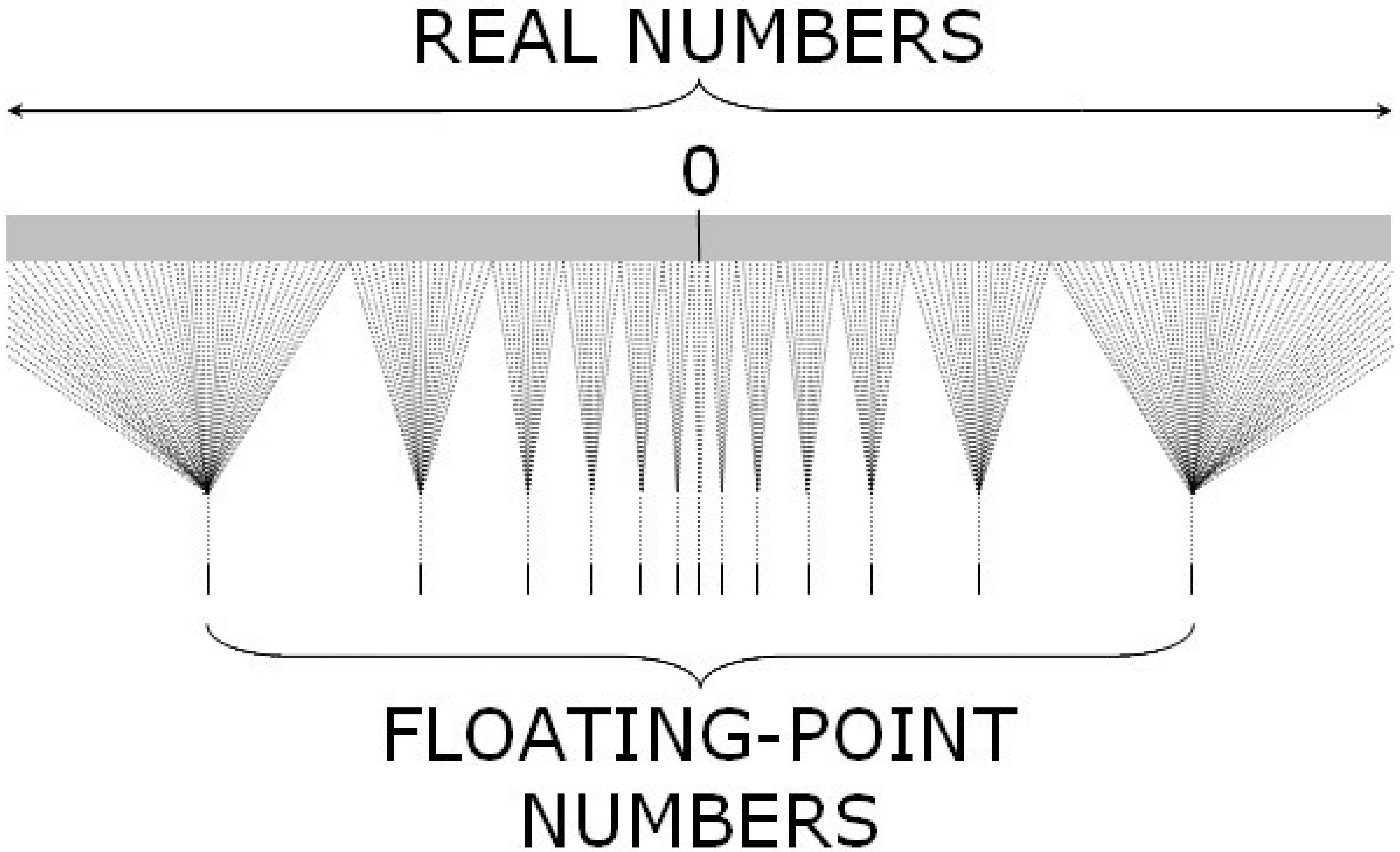
Per rappresentare un numero razionale* (al netto degli errori di approssimazione) vi sono due strade principali:

Notazione in virgola fissa: dedico un numero di cifre alla parte intera e un numero di cifre alla parte decimale:

$$\pm \boldsymbol{iiii, ddd}$$

Notazione in virgola mobile (floating point – IEEE 754): faccio scorrere la virgola secondo le esigenze di rappresentazione – rappresenta molti più valori utilizzando gli stessi bit rispetto alla virgola fissa

** I numeri reali sono composti dai numeri razionali + i numeri irrazionali: i numeri irrazionali non sono rappresentabili*


REAL NUMBERS
0
FLOATING-POINT
NUMBERS

Rappresentazione in mantissa, base ed esponente

Qualsiasi numero può essere scritto nella seguente forma:

$$\pm M \times b^{\pm e}$$

Dove M prende il nome di **mantissa**, b è la **base** ed e è l'**esponente**

Quando tale sistema viene applicato alla base 10 prende il nome di *notazione scientifica* (ad esempio $0{,}83234 \times 10^2$)

Naturalmente, tale rappresentazione dovrà essere approssimata destinando un certo numero di bit alla mantissa e un certo numero di bit all'esponente

Rappresentazione floating point

Precisione nel floating point:

	Precisione singola	Precisione doppia
Bit per il segno	1	1
Bit per l'esponente	8	11
Bit per la mantissa	23	52
Bit totali	**32**	**64**
Intervallo esponente	[-126, 127]	[-1022, 1023]
Bias esponente	127	1023

Non è necessario memorizzare la base in quanto è implicita (2)

Non tutte le configurazioni di esponenti sono disponibili, alcune sono riservate

Gli esponenti sono rappresentati in **forma polarizzata**, ovvero si memorizza in binario l'esponente sommato a una costante che è detta **bias** – ciò consente di effettuare più facilmente controlli di maggioranza o minoranza tra interi polarizzati (l'esponente più basso assume valore 00000000 e il più alto 11111111)

Rappresentazione in floating point

Vi sono diverse tipologie di numeri nel floating point:

segno	esponente	mantissa	
$\pm$	$\neq 0$ e $\neq 111\ldots111$	Qualsiasi	Numero normalizzato
$\pm$	0	Qualsiasi (tranne 0)	Numero denormalizzato
$\pm$	0	0	± 0
$\pm$	$111\ldots111$	0	$\pm\infty$
$\pm$	$111\ldots111$	Qualsiasi (tranne 0)	NaN (Not a Number)

Floating point: numeri normalizzati

Un numero **normalizzato** espresso in floating point su un calcolatore è definito come segue:

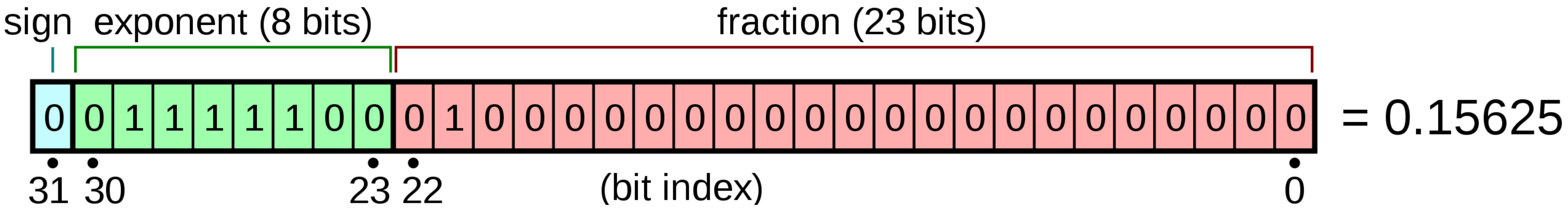
$$\pm 1,xxxxxxx_2 \times 2^{yyyy}$$

$xxxxxxx$ sono i bit destinati alla mantissa e $yyyy$ i bit destinati all'esponente.

Il valore di mantissa in un numero normalizzato è sempre compreso tra 1 (compreso) e 2 (escluso)

Si usano tutti i bit x per identificare la parte frazionaria (l'1 intero è implicito)

Floating point: numeri normalizzati



Il valore di un numero a 32 bit in floating point è dato dalla seguente formula:

$$(-1)^{b_{31}} \times 2^{(b_{30}, b_{29}, \dots, b_{23})-127} \times (1, b_{22}b_{21} \dots b_0)_2$$

Convertire un numero reale in binario

Per la parte intera si procede come già visto

Per la parte frazionaria si moltiplica il valore per 2 e si prende la cifra intera ricavata, la si sottrae e si procede fin quando non si esaurisce la precisione (numero di cifre binarie che è possibile memorizzare) o il risultato è 1

Esempio:

$$19{,}3125_{10} = ?_2$$

Step 1: Parte intera $19_{10} = 10011_2$

Step 2: Parte decimale

$$0{,}3125 \times 2 = 0{,}625$$

$$0{,}625 \times 2 = 1{,}250$$

$$0{,}250 \times 2 = 0{,}500$$

$$0{,}500 \times 2 = 1{,}0 \longrightarrow$$

Ho ottenuto precisamente 1
FINE ALGORITMO

Quindi $19{,}3125_{10} = 10011{,}0101_2$

Convertire un numero reale in binario

Proviamo a vedere se è vero…

2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}
1	0	0	1	1	,	0	1	0	1
16			2	1			0,25		0,0625

$$16 + 2 + 1 + 0{,}25 + 0{,}0625 = 19{,}3125$$

Dopo aver effettuato la conversione si imposta l'esponente in maniera tale da far scorrere la virgola del numero di posizioni necessarie per rappresentare il numero correttamente

Floating point: numeri denormalizzati

Un numero **denormalizzato** espresso in floating point su un calcolatore è definito come segue:

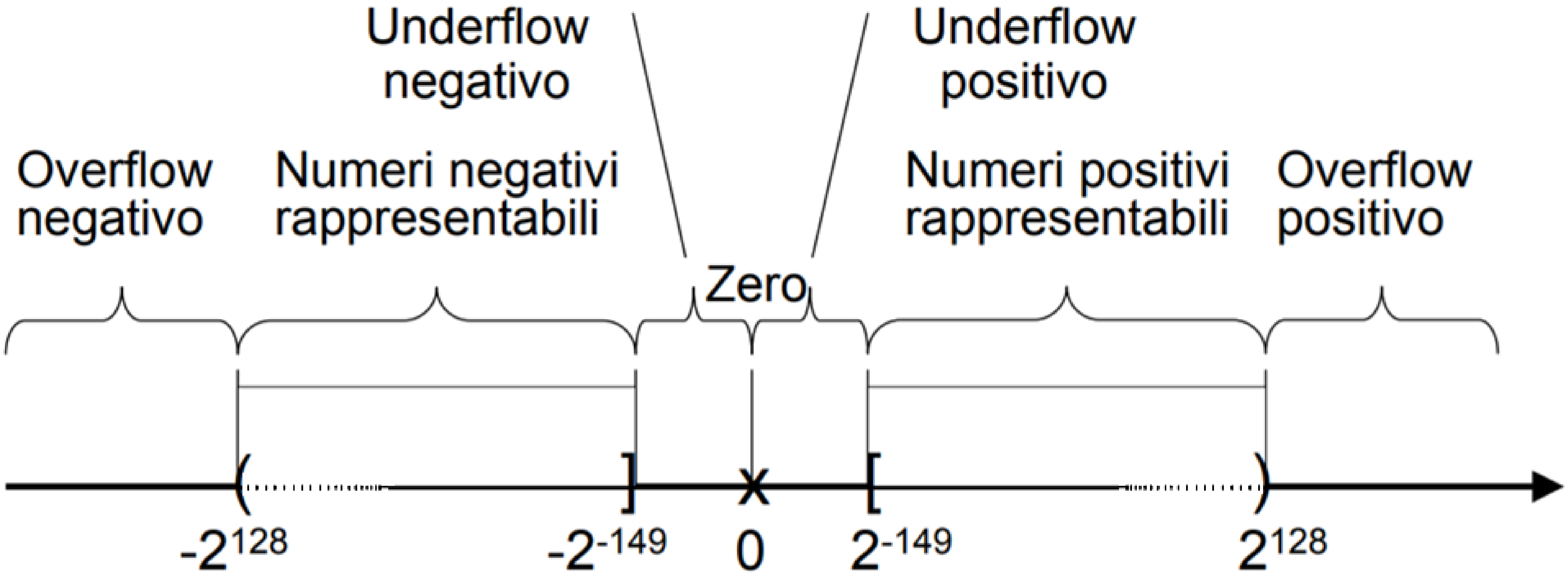
$$\pm 0, xxxxxxx_2 \times 2^{1-b}$$

b è il bias

Qui la mantissa è sempre compresa tra 0 e 1 e i bit dell'esponente sono impostati a 0

Servono a riempire l'intervallo tra lo zero e il più piccolo numero normalizzato rappresentabile

Floating point su 32 bit (precisione singola)



Rappresentazione in floating point

In generale l’aritmetica a virgola mobile è affetta da alcune problematiche:

- Non sono valide in generale la proprietà associativa e la proprietà distributiva
- Assorbimento: ad esempio $10^{15} + 1 = 10^{15}$
- Cancellazione: si ottiene quando sottraendo due numeri molto vicini si ottiene 0
- Arrotondamento

Gli errori di calcolo sono invece ottenuti da:

- Le operazioni in overflow danno risultato $+\infty$, $-\infty$
- Situazioni di underflow, ovvero valori molto piccoli trasformati in 0
- Le operazioni impossibili (ad esempio la radice quadrata di un numero negativo) restituisce NaN

Rappresentazione in floating point

I problemi di arrotondamento sono dati da una duplice natura:

1. Operazioni aritmetiche: ${}^{2}/_{3} = 0{,}666667$
2. Numeri non rappresentabili

Ad esempio $0{,}1$ non è un numero rappresentabile:

$$
\begin{aligned}
0{,}1 \times 2 &= 0{,}2 \\
0{,}2 \times 2 &= 0{,}4 \\
0{,}4 \times 2 &= 0{,}8 \\
0{,}8 \times 2 &= 1{,}6 \\
0{,}6 \times 2 &= 1{,}2 \\
0{,}2 \times 2 &= 0{,}4 \\
0{,}4 \times 2 &= 0{,}8 \\
&\dots
\end{aligned}
$$

Si ha un evidente ciclo infinito sulla cifra finale 2-4-8-6

Operazioni in floating point

Confronto di uguaglianza: Poiché i dati possono provenire da operazioni di natura diversa ha senso definire l'uguaglianza come segue:

$$\mathrm{A} = \mathrm{B} \iff |A - B| < \varepsilon$$

ovvero se i due numeri sono «sufficientemente» vicini tra loro

Confronto maggiore/minore: non è un caso che vengano memorizzati nell'ordine segno, esponente e mantissa. Per confrontarli è sufficiente scorrere i bit dei due numeri fin quando non si trova un bit diverso:

- Se si trova nel segno, è più grande il numero con il segno positivo (0)
- Se si trova nell'esponente o nella mantissa è più grande il numero con il bit a 1

Operazioni in floating point

Somma/sottrazione:

- Si allineano i due numeri per raggiungere lo stesso esponente
- Si sommano le mantisse
- Si normalizza il risultato
- Si controlla se è overflow o underflow
- Si arrotonda il numero
- Se non è normalizzato, lo si normalizza

Prodotto/divisione:

- Si sommano gli esponenti – bias
- Si moltiplicano le mantisse
- Si normalizza il risultato
- Si controlla se è overflow o underflow
- Si arrotonda il numero
- Se non è normalizzato, lo si normalizza