Rappresentazione per segno e modulo

- Algoritmo per le operazioni di somma e sottrazione complesso
  1. Confrontare i bit di segno dei due numeri
  2. Se i bit di segno sono uguali
     - Il risultato ha lo stesso segno ed il valore è dato dalla somma dei valori assoluti dei due numeri
  3. Se i bit di segno sono uguali
     - Si confrontano i valori assoluti dei due numeri
     - Il risultato ha il segno del numero maggiore in valore assoluto ed ha valore pari alla differenza fra i due numeri

N=3bit
000 →+0
001 →+1
010 →+2
011 →+3
100 →-0
101 →-1
110 →-2
111 →-3

Somma (-2)+(-1) = -3
110+101=011 con riporto 1

Somma (-3)+(+2) = -1
111+010=001 con riporto 1

Rappresentazione per complemento alla base b

- Per semplificare le operazioni aritmetiche sui numeri interi si adotta una rappresentazione dei numeri detta ***in complemento alla base***.

- ***Definizione di Complemento alla Base:***
  - Dato un numero x rappresentato con m cifre, si definisce il complemento alla ***base*** (b) di x, C(x), come segue:

$$C(x) = b^m - x$$

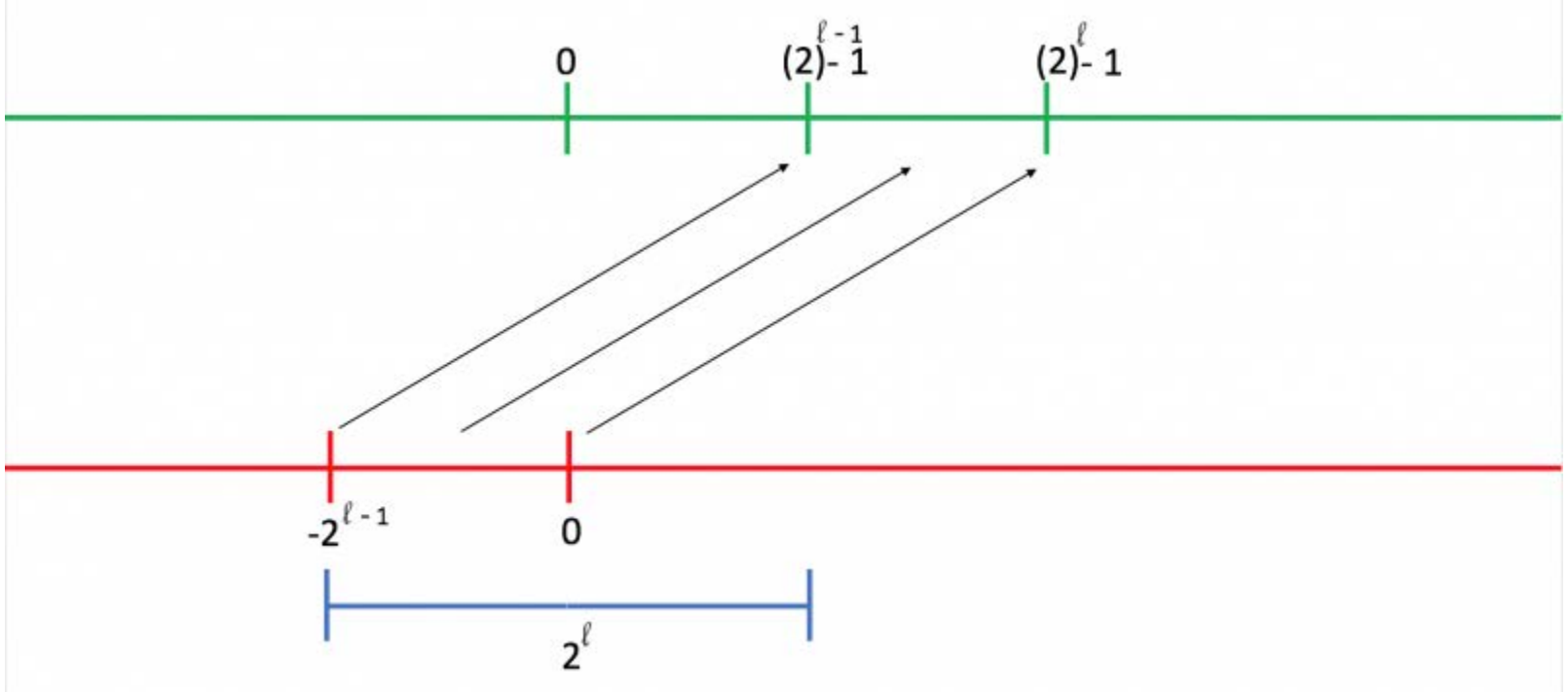
(con m=1)

x'=(b-1)-x (complemento alla base b diminuito di x)

x''=b-x → x''=b-1-x+1=x'+1 →(complemento alla base b di x)

Rappresentazione per complemento a 2

RAPP	INT	CL2
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1



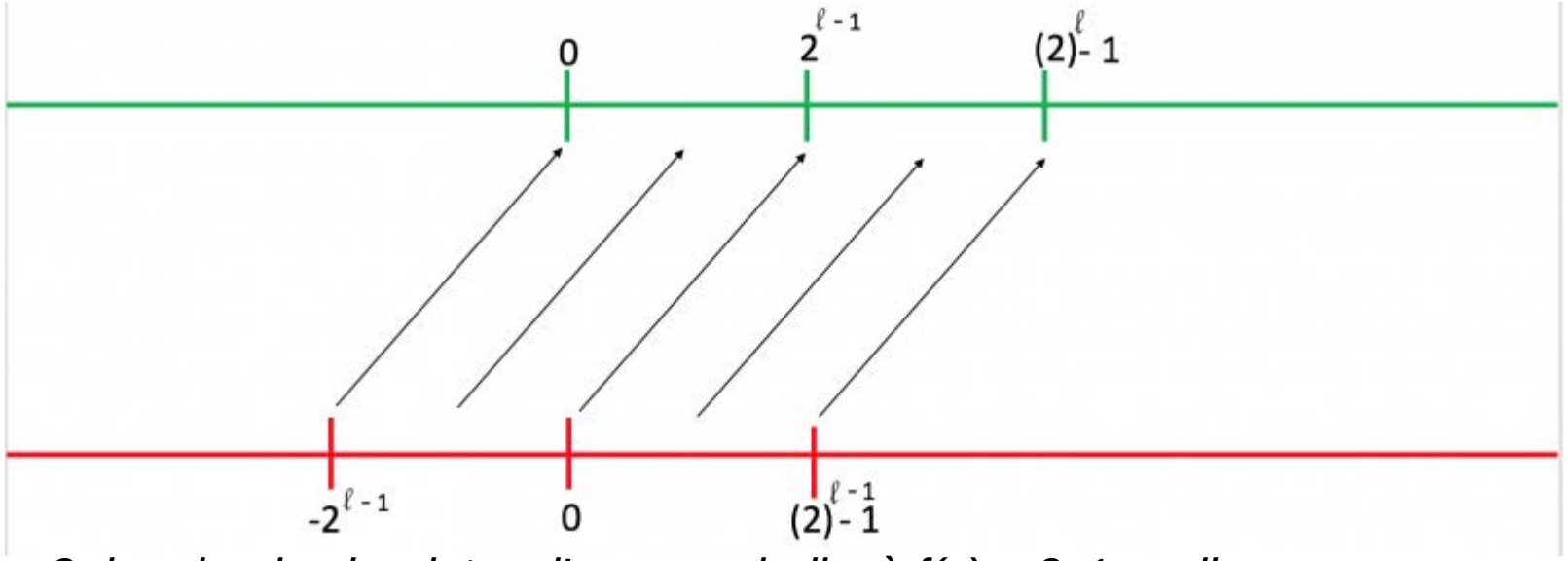
n=3 ricordando che dato x il complemento a 2 è $f(x) = 2^n - x$ allora per x negativo si ha:
$f(-4) = 2^3 - |4| = 4$ ovvero 100
$f(-3) = 2^3 - |3| = 5$ ovvero 101
$f(-2) = 2^3 - |2| = 6$ ovvero 110
$f(-1) = 2^3 - |1| = 7$ ovvero 111

Rappresentazione per eccessi

- Nella *rappresentazione per eccesso* i numeri negativi si determinano come somma di se stessi e K con $K=2^{l-1}$ dove **l** è il numero di bit utilizzati.
- Si noti che il sistema è identico al complemento a due con il bit di segno invertito.
- I numeri compresi in $[-2^{l-1}, 2^{l-1}-1]$ sono *mappati* nell'intervallo $[0, 2^{l}-1]$
- In tale rappresentazione, il numero binario che rappresenta -2^{l-1} sarà associato allo zero, mentre i valori minori di 2^{l-1} ai numeri negativi e quelli maggiori a quelli positivi
- Nel caso di **l= 8** i numeri appartenenti a **[–128, 127]** sono mappati nell'intervallo **[0, 255]** (con i numeri da 0 a 127 considerati negativi, il valore 128 corrisponde allo 0 e quelli maggiori di 128 sono positivi).

Rappresentazione per eccessi

RAPP	INT	ECC K
000	0	-4
001	1	-3
010	2	-2
011	3	-1
100	4	0
101	5	+1
110	6	+2
111	7	+3



n=3 ricordando che dato x l'eccesso k di x è $f(x) = 2^{n-1}+x$ allora per x negativo si ha

$f(-4) = 2^{3-1} - 4 = 0$ ovvero 000
$f(-3) = 2^{3-1} - 3 = 1$ ovvero 001
$f(-2) = 2^{3-1} - 2 = 2$ ovvero 010
$f(-1) = 2^{3-1} - 1 = 3$ ovvero 011

Rappresentazione dei numeri reali

- Devono essere rappresentate le tre parti componenti del numero, ossia:

  - ***Segno***
    - Tipica informazione binaria (rappresentabile con un solo bit)
  - ***Stringa di Cifre***
  - ***Posizione della Virgola***

Stringa di Cifre

- Costituisce il ***modulo*** del numero ed è determinata dalla aritmetica adottata.

- Deve essere finita e di lunghezza predeterminata (la si completa con 0 a sinistra)

  00341 10000 14511 00001

- Dicesi ***significativa*** la sottostringa ottenuta dalla stringa depennando gli 0 a sinistra

  341 10000 14511 1

Esempi di rappresentazione in virgola fissa

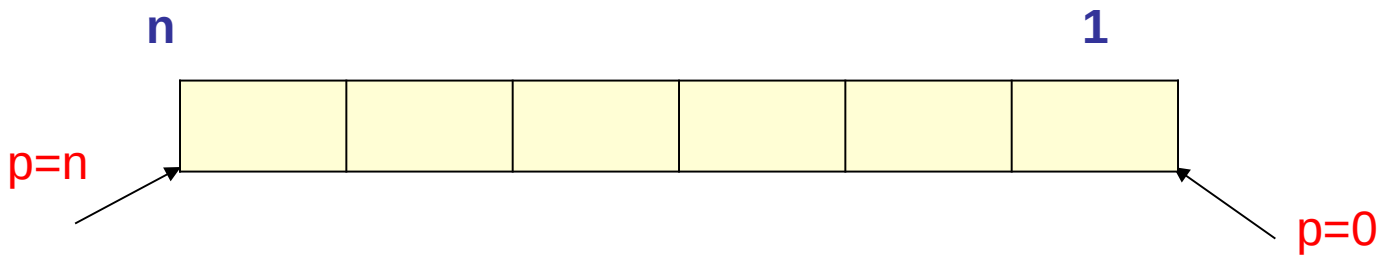
- **Valori da rappresentare:**
  3491 0349 0035 0003

- Se la posizione del punto è in prima posizione:
  0.3491 0.0349 0.0035 0.0003

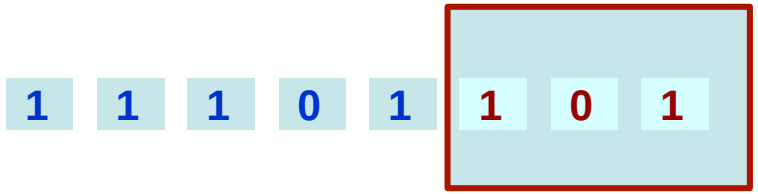
- Se la posizione del punto è in ultima posizione: 3491
  349 35 3

Rappresentazione in virgola fissa

- Su n bit sia p la posizione del punto frazionario
  - Se p=0 numero intero
  - Se p=n numero frazionario puro (≥ 0 e <1)



Rappresentazione dei numeri reali con virgola fissa



N =8 b=2 p=3
Numero rappresentato
11101.101

Max valore rappresentabile :

1 1 1 1 1 1 1 1

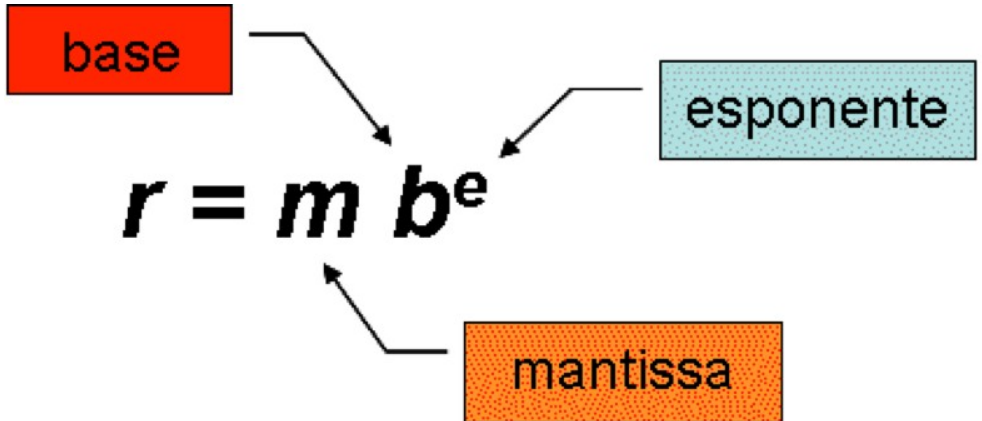
$1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 + 1*2^{-1} + 1*2^{-2} + 1*2^{-3}$

16 + 8 + 4 + 2 + 1 + 0.50 + 0.25 + 0.125 = **31.875**

Va bene questa rappresentazione?

Rappresentazione in virgola mobile

- Detta anche floating point
- I numeri reali r vengono rappresentati in binario attraverso la seguente notazione scientifica
- L'esponente determina l'ampiezza dell'intervallo di valori preso in considerazione, mentre il numero di cifre della mantissa determina la precisione del numero (ossia con quante cifre significative sarà rappresentato)



Notazione Scientifica

- Viene adottata per:
  - la sua indipendenza dalla posizione della virgola;
  - la possibilità di rappresentare con poche cifre numeri molto grandi oppure estremamente piccoli;

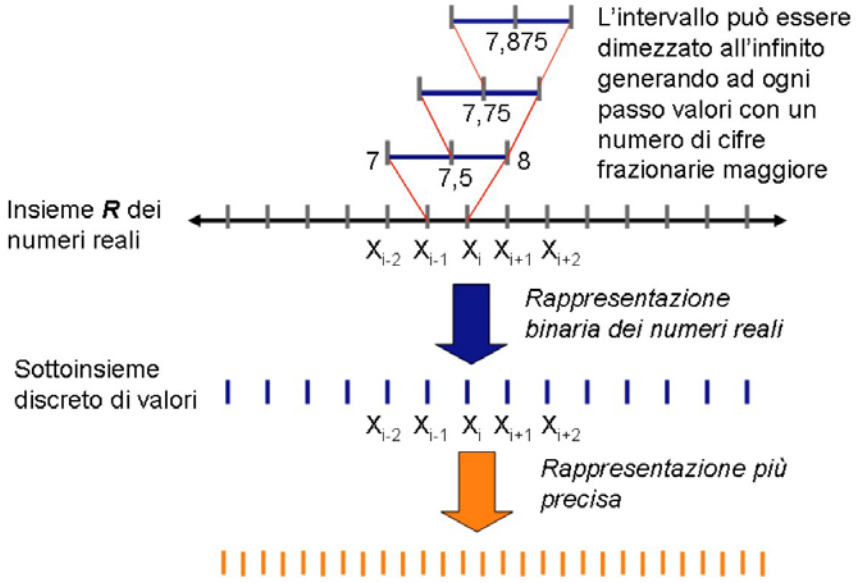
149 000 000 000 → **$1.49 * 10^{11}$**

149 000 000 000 → **$14.9 * 10^{10}$**

  - la possibilità di trascurare tutti gli zeri che precedono la prima cifra significativa con la normalizzazione della mantissa;

Rappresentazione finita e discreta dei numeri reali

- In un intervallo reale, comunque piccolo, esistono infiniti valori (i numeri reali formano un continuo)
- I valori rappresentabili in binario appartengono invece ad un sottoinsieme che contiene un numero finito di valori reali ognuno dei quali rappresenta un intervallo del continuo.
  - Diviso l'insieme dei numeri reali in intervalli di fissata dimensione si ha che ogni x nell'intervallo $[x_i, x_{i+1}[$, viene rappresentato con x_i



Effetti delle approssimazioni

- La rappresentazione di un numero reale *x* con il valore *X* rappresentante l'intervallo a cui *x* appartiene impone notevoli problemi di approssimazione.
- L'errore assoluto che si commette rappresentando *x* con *X* è:
  - **$| x - X |$**
  - $X = X_i$ – approssimazione per difetto
  - $X = X_{i+1}$ – approssimazione per eccesso
- L'errore relativo che si commette è:
  - **$| x - X| / |x|$**

Esempi errore di arrotondamento

NUMERO	ARROTONDAMENTO	ERRORE
0,00347	0,0035	$3*10^{-5} = 0.3 *10^{-4}$
0,000348	0,0003	$48*10^{-6} = 0.48*10^{-4}$
0,00987	0,0099	$3*10^{-5} = 0.3 *10^{-4}$
0,000987	0,0010	$13*10^{-6} = 0.13*10^{-4}$

- Per un aritmetica a quattro cifre decimali
  - Con un errore massimo sull'ultima cifra di $(0.5 * 10^{-4})$
- In generale se -m è il peso della cifra meno significativa, l'errore massimo relativo che si commette è:

- $\frac{1}{2} \times 10^{-m}$

Normalizzazione

- Per ogni numero esistono infinite coppie che lo rappresentano. Esempio (*b*= 10):

  346.09801 è rappresentato da

  (346.09801, 0) oppure

  (346098.01, -3) oppure

  (0.034609801, 4) … ecc.

- Se si fissa la posizione della virgola rispetto alla prima *cifra significativa*
  - Unico rappresentante: (3.4609801, 2)
- La rappresentazione in virgola mobile si dice ***normalizzata*** se la scelta della coppia (Mantissa, esponente) è tale che la prima cifra della mantissa sia diversa da zero 0

Esempio di Normalizzazione

- rappresentazione con b = 10,
- cinque cifre per la mantissa considerata minore di 10,
- due cifre per l'esponente,
- rappresentazione normalizzata con la prima cifra diversa da zero;
- si hanno le seguenti rappresentazioni normalizzate:

$0{,}384 \rightarrow 0{,}3840 \times 10^{0}$
$1345 \rightarrow 0{,}1345 \times 10^{4}$
$333 \rightarrow 0{,}3330 \times 10^{3}$
$0{,}000001 \rightarrow 0{,}1000 \times 10^{-5}$

… condizione di overflow quando

$$x > 0{,}9999 \times 10^{99}$$

e di underflow quando

$$x < 0{,}1000 \times 10^{-99}$$

Overflow e Underflow

- L'insieme $\mathcal{R}$ è costituito da infiniti valori nell'intervallo $]-\infty, +\infty[$
- I numeri reali rappresentabili sono invece definiti in un insieme limitato con estremi predefiniti $]minreal, maxreal[$
  - **Overflow**: condizione che si verifica quando i valori o sono più piccoli di minreal o più grandi di maxreal;
    - la divisione per valori molto piccoli può facilmente superare il valore di Overflow
  - **Underflow**: condizione che si verifica quando un valore, per effetto delle approssimazioni, viene confuso con lo zero.
    - la divisione per valori molto grandi può facilmente superare il valore di Underflow

Overflow e Underflow

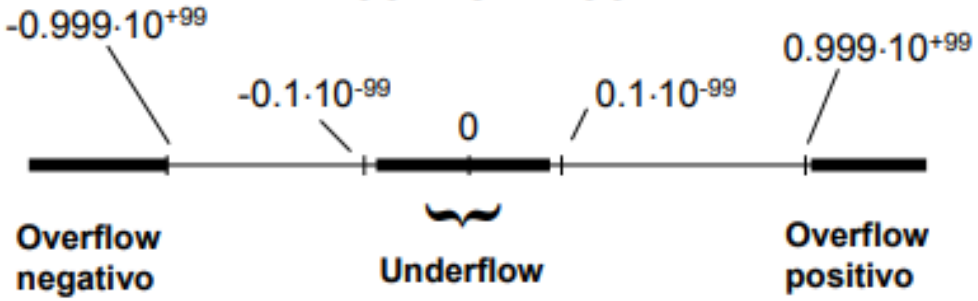
Esempio in base 10

- Numerali a 5 cifre $\pm$ .XXX $\pm$ EE
- Mantissa: 3 cifre con segno

$$0.1 \leq |m| < 1$$

- Esponente: 2 cifre con segno

$$-99 \leq e \leq +99$$



Osservazione

- Gli intervalli $[x_i, x_{i+1}]$ non hanno tutti la stessa ampiezza a causa del numero finito di cifre della mantissa:
  - più ci si avvicina alla condizione di Overflow gli intervalli si fanno sempre più ampi,
  - più ci si avvicina alla condizione di Underflow gli intervalli si fanno sempre più piccoli.

- Con l'esempio precedente è facile osservare il fenomeno confrontando gli intervalli:
  - $[0.1000 \times 10^{-99}, 0.1001 \times 10^{-99}]$
  - $[0.1000 \times 10^{99}, 0.1001 \times 10^{99}]$

Osservazione

- Il numero di cifre m della mantissa determina la precisione con cui i numeri sono rappresentati

- L'errore assoluto che si commette nella rappresentazione di un numero reale è variabile.
  - Dipende dall'intervallo di rappresentazione

- L'errore relativo è limitato superiormente da

$$\frac{1}{2} \times 10^{-m}$$

Operazioni in virgola mobile 1/2

- Le operazioni possono generare errori di approssimazione:
  - Somma e sottrazione richiedono l'allineamento degli esponenti
    - $100x10^{0} + 100x10^{-2} = 100x10^{0} + 1x10^{0} = 101x10^{0}$
  - Prodotto e divisione richiedono operazioni separate per mantisse ed esponenti
    - $100x10^{0} * 100x10^{-2} = (100*100) \times 10^{(0-2)}$
- L'allineamento degli esponenti può produrre come effetto indesiderato la scomparsa di alcune cifre
  - $1.9099x10^{0}+5.9009x10^{4}$ diventa $0.0001x10^{4}+5.9009x10^{4}$
  - Con il troncamento delle cifre 9099 del numero ad esponente minore

Se c'era 10^{5} ?

E Possiamo allineare al più piccolo?

Operazioni in virgola mobile 2/2

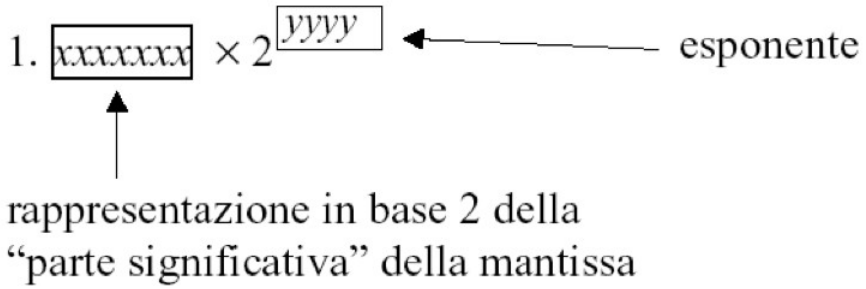
- Se si sottraggono numeri di valori quasi uguale, le cifre più significative si eliminano tra loro e la differenza risultante perde un certo numero di cifre significative, o anche tutte (cancellazione)

- La divisione per valori troppo piccoli può superare la condizione di Overflow

- La divisione per valori troppo grandi può superare la condizione di Underflow

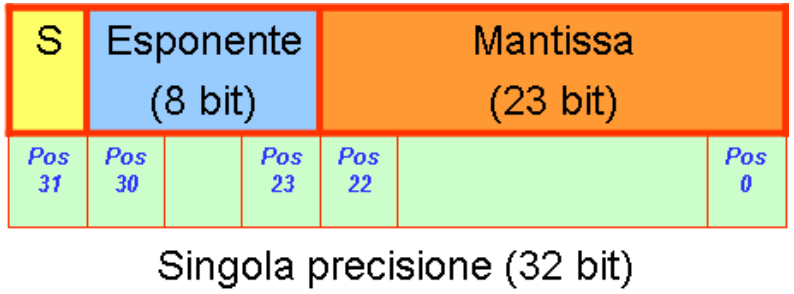
Vantaggi della rappresentazione in virgola mobile

- Se si conviene che le mantisse siano trasformate in valori minori di 10 con operazioni interne, un numero reale può essere rappresentato nella memoria di un calcolatore con un numero intero indicativo della parte decimale della mantissa e con un altro numero intero per l'esponente.
- Per esempio il numero viene rappresentato con la coppia di numeri interi (1230, -9) e gestito con operazioni interne che ne costruiscono l'effettivo valore.
- Analogamente in base due

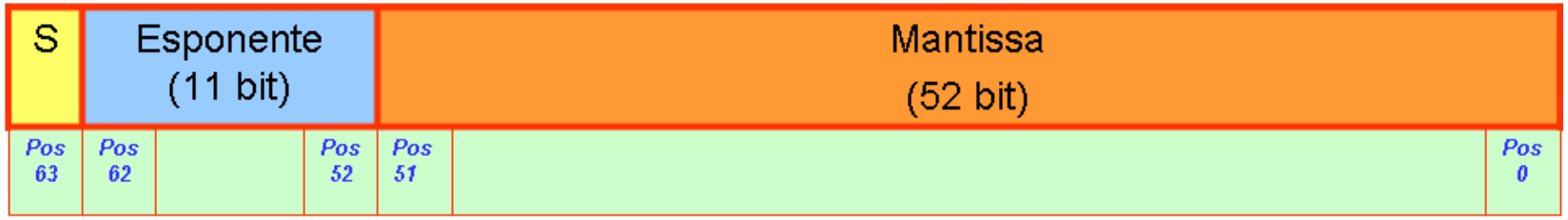


Standard virgola mobile

- Standard 754 IEEE (Institute of Electrical and Electronics Engineers): definisce principalmente tre formati numerici a virgola mobile:
  - singola precisione o precisione semplice (32 bit),
  - doppia precisione 64 bit),
  - precisione estesa (80 bit).



Singola precisione (32 bit)



Doppia precisione (64 bit)

Rappresentazioni VM precisione singola e doppia

- 1 bit per il segno del numero complessivo, (0 per positivo ed 1 per negativo);
- 8 bit nel caso della singola precisione (11 per la doppia precisione) per l'**esponente rappresentato per eccesso 127:**
- 23 bit nel caso della singola precisione (52 per la doppia) per la mantissa.
- La mantissa è normalizzata per cui comincia sempre con un 1 seguito da una virgola binaria, e poi a seguire il resto delle cifre.
  - Lo standard prevede l'assenza sia del primo bit che del bit della virgola perché sono sempre presenti:

Argomento	Precisione singola 32 Bit	Precisione doppia 64 bit
Bit del segno	1	1
Bit per l'esponente	8	11
Bit per la mantissa	23	52
Cifre decimali mantissa	Circa 7 (23/3.3)	Circa 15 (52/3.3)
Esponente (rappresentazione)	base 2 ad eccesso 127	base 2 ad eccesso 1023
Esponente (valori)	[-126, 127]	[-1022, 1023]

IEEE 754 a 32 bit: esponente

- Esponente (8bit)
  - Rappresentato **in eccesso 127** (polarizzazione o bias)
  - L'intervallo di rappresentazione è [-127, 128]

    $[-k , 2^n-1-k]$ → che diventa $[-2^{n-1},2^{n-1}-1]$ per $k=2^{n-1}$
    - -127 si codifica con -127+127=0 -> 00000000
    - 0 si codifica con 0+127=127 -> 01111111
    - 128 si codifica con 128+127=255 -> 11111111
  - Le due configurazioni estreme sono riservate
    - gli esponenti possono assumere solo i valori in [-126, 127], corrispondenti alle configurazioni [1,254]

Esempio #1 (1/2)

- Codifica del valore N reale con rappresentazione in virgola mobile nel formato *singola precisione*
- 1 bit segno, 8 bit esponente, 23 bit mantissa
- N = 73.625
- 73 e' 1 0 0 1 0 0 1
- Conversione Parte frazionaria:
  - .625 * 2 = 1.250
  - .250 * 2 = 0.5
  - .5 * 2 = 1.0

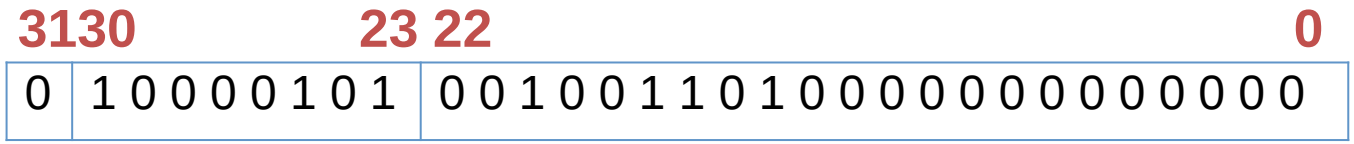
1 0 0 1 0 0 1. 1 0 1

- Normalizzazione (6 posizioni a sinistra esponente +6)

1.0 0 1 0 0 1 1 0 1

Esempio #1 (2/2)

- Mantissa: 1.0 0 1 0 0 1 1 0 1→ primo bit lo ignoriamo
- Valore esponente da definire ad eccesso 127
- Valore esponente 127 + 6 =133 = 10000101



Esempio #2

- 1 10000001 01000000000000000000000
  - Segno negativo;
  - Esponente: $2^7+2^0=129$
    - 129 -127=2
  - Mantissa: $1+2^{-2}=1.25$
  - Numero: $-1.25 \times 2^2=-5$

Esempio #3

- 8.5
- Segno +
- 8.5 in binario: 1000 .1=1.0001 2^3
  - Mantissa: 00010000000000000000000
  - Esponente: 3+127=130=10000010
  - **NUMERO: 0 10000010 00010000000000000000000**

Configurazioni particolari

- **Mantissa 0 e esponente 0**: rappresentano 0
- **Mantissa 0 ed esponente 255:** rappresentano infinito
- **Mantissa diversa da 0 e esponente 255**: rappresentato la situazione di **Not a Number** (NAN), cioè un valore indefinito (esempio il risultato di una divisione per 0 o la radice quadrata di un numero negativo)
- **Mantissa diversa da 0 e esponente 0**: rappresentano numeri denormalizzati

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

Da analogico a digitale premessa

- Tutti i dispositivi di un elaboratore sono realizzati con **tecnologia digitale**
- Dati ed operazioni sono codificati mediante sequenze di bit (cifre binarie “0” ed “1”)

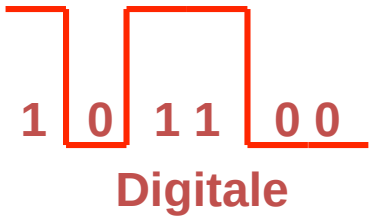
- Con queste due sole cifre, usate in combinazioni diverse, è possibile rappresentare tutti i dati, siano questi *parole*, *numeri*, *immagini* o *filmati*, *suoni*.

Segnali Analogici e Digitali

- Oggi, con l'evoluzione tecnologica, molte informazioni nascono già in formato digitale (v. es. macchine fotografiche). Per molte altre è necessario procedere alla conversione da analogico a digitale.

- Un segnale è ***analogico*** quando i valori utili che lo rappresentano sono continui (*infiniti*) in un intervallo di tempo.

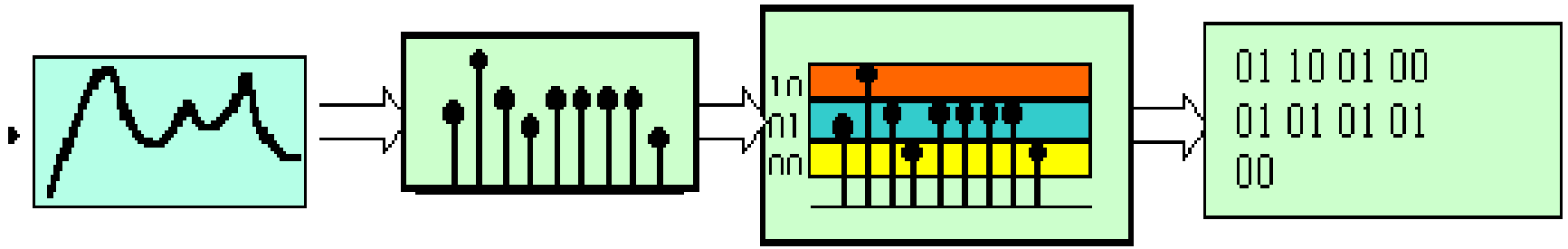
- Un segnale è ***digitale*** quando i valori utili che lo rappresentano sono in formato discreto (non variano in modo continuo ma solo per gradini,senza possibilità di valori intermedi tra due gradini successivi).



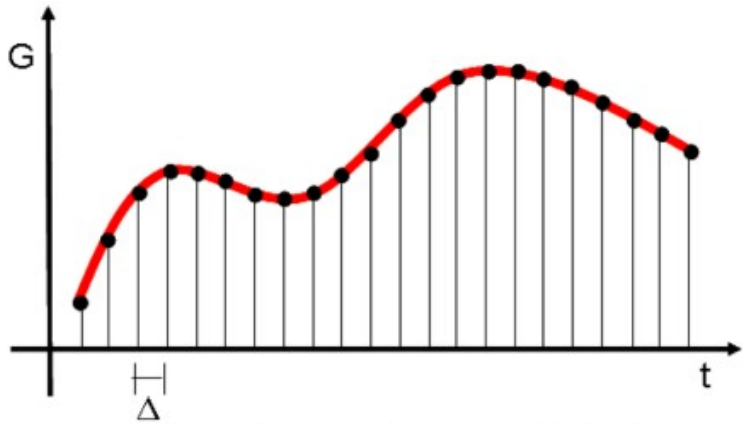
Da analogico a digitale

- La trasformazione da analogico a digitale si realizza per mezzo una operazione detta *campionamento ed una di quantizzazione*:
  - **Campionamento**: a intervalli regolari di tempo, si va a osservare quali valori assume la funzione analogica e se ne conservano le osservazioni o *campioni*
  - **Quantizzazione**: l'operazione di *quantizzazione* approssima *i campioni* ad un certo numero prefissato di livelli

Convertitori Analogici-Digitali



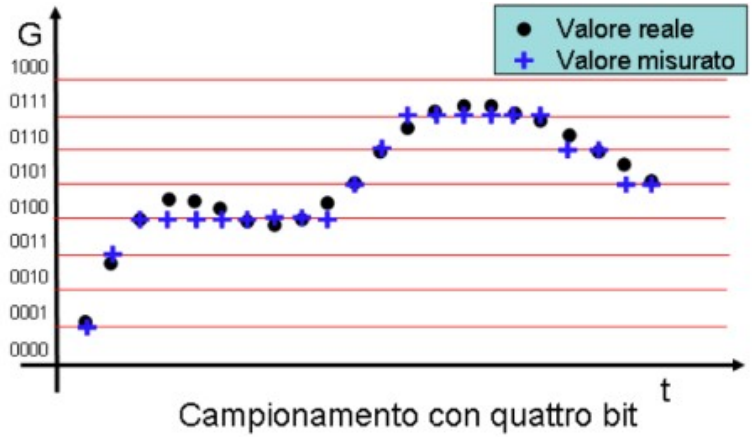
Campionamento



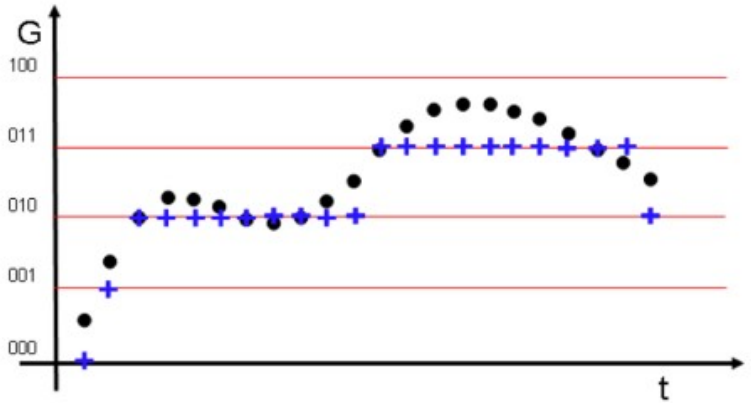
Campionamento con periodo Δ
della grandezza G funzione del tempo

Campionamento meno preciso
con periodo 2Δ della stessa grandezza

Quantizzazione



Campionamento con quattro bit



Campionamento con tre bit
(più approssimato)

$G(t_1)$	0001	000
$G(t_2)$	0011	001
$G(t_3)$	0100	010
$G(t_4)$	0100	010
$G(t_5)$	0100	010
$G(t_6)$	0100	010
$G(t_7)$	0100	010
$G(t_8)$	0100	010
$G(t_9)$	0100	010
$G(t_{10})$	0100	010
$G(t_{11})$	0101	010
$G(t_{12})$	0110	011
$G(t_{13})$	0111	011
$G(t_{14})$	0111	011
$G(t_{15})$	0111	011
$G(t_{16})$	0111	011
$G(t_{17})$	0111	011
$G(t_{18})$	0111	011
$G(t_{19})$	0110	011
$G(t_{20})$	0110	011
$G(t_{21})$	0101	011
$G(t_{22})$	0101	010

Rappresentazione Testi

Il **testo** è uno degli oggetti digitali più diffuso nel mondo informatico. Molte sono le applicazioni che generano e manipolano i cosiddetti documenti elettronici.

Un **testo digitale** è una stringa di simboli ad ognuno dei quali viene associato un codice binario secondo un prefissato standard.

Rappresentazione di caratteri

- La necessità di elaborare caratteri si presenta in moltissimi contesti
- Quando parliamo di caratteri ci riferiamo a :
  - Lettere maiuscole e minuscole dell'alfabeto inglese
  - Cifre decimali
  - Caratteri di interpunzione (!, ?, .,..)
  - Altri caratteri (più, meno, underscore..)
- Per un totale di quasi 100 caratteri (ci vogliono 7 bit)
- Esistono varie possibilità di costruire la tabella di rappresentazione dei caratteri
- l'ente di standardizzazione americano ANSI ha definito nel 1968 la tabella ASCII (American Standard Code for Information Interchange) usata come standard

ANSI ASCII

- Alla fine degli anni sessanta l'ente americano di standardizzazione ANSI (American National Standards Institute) decise di fissare un alfabeto che consentisse a tutti i calcolatori, anche di produttori diversi, di poter comunicare tra loro o con i dispositivi ad essi collegati.
- I simboli dell'alfabeto vennero elencati in una tabella per codificare, vari tipi di caratteri: **alfabetici**, **numerici**, di **punteggiatura**, **simbolici**, e anche alcuni codici da usare come **controllo** della comunicazione tra una macchina e l'altra (per esempio, per segnalare l'inizio o la fine di una trasmissione).
- La tabella fu chiamata ***ASCII***, ossia *American Standard Code for Information Interchange*.

- ASCII standard
  - Codifica 128 simboli utilizzando 7 bit ($2^7=128$)
- ASCII esteso
  - Codifica 256 simboli utilizzando un byte per carattere ($2^8=256$)

Codice ASCII a 7 bit

	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP 32	0 48	@ 64	P 80	` 96	p 112
0001	SOH	DC1	! 33	1 49	A 65	Q 81	a 97	q 113
0010	STX	DC2	« 34	2 50	B 66	R 82	b 98	r 114
0011	ETX	DC3	# 35	3 51	C 67	S 83	c 99	s 115
0100	EOT	DC4	$ 36	4 52	D 68	T 84	d 100	t 116
0101	ENQ	NAK	% 37	5 53	E 69	U 85	e 101	u 117
0110	ACK	SYN	& 38	6 54	F 70	V 86	f 102	v 118
0111	BEL	ETB	’ 39	7 55	G 71	W 87	g 103	w 119
1000	BS	CAN	(40	8 56	H 72	X 88	h 104	x 120
1001	HT	EM	) 41	9 57	I 73	Y 89	i 105	y 121
1010	LF	SUB	* 42	: 58	J 74	Z 90	j 106	z 122
1011	VT	ESC	+ 43	; 59	K 75	[91	k 107	{ 123
1100	FF	FS	, 44	< 60	L 76	\ 92	l 108	| 124
1101	CR	GS	− 45	= 61	M 77	] 93	m 109	} 125
1110	SO	RS	. 46	> 62	N 78	^ 94	n 110	~ 126
1111	SI	US	/ 47	? 63	O 79	_ 95	o 111	DEL

Esempi:

A= 100 0001
ossia 65

a= 110 001
ossia 97

C= 100 0011
ossia 67

Il codice binario si ottiene affiancando i 3 bit della colonna e poi i 4 della riga

Codice ASCII a 8 bit (1/2)

carattere	codice ASCII	equivalente esadecimale	equivalente numerico	carattere	codice ASCII	equivalente esadecimale	equivalente numerico
NUL	00000000	00	0	*spazio*	00100000	20	32
SOH	00000001	01	1	!	00100001	21	33
STX	00000010	02	2	”	00100010	22	34
ETX	00000011	03	3	#	00100011	23	35
EOT	00000100	04	4	$	00100100	24	36
ENQ	00000101	05	5	%	00100101	25	37
ACK	00000110	06	6	&	00100110	26	38
BEL	00000111	07	7	’	00100111	27	39
BS	00001000	08	8	(	00101000	28	40
TAB	00001001	09	9	)	00101001	29	41
LF	00001010	0A	10	*	00101010	2A	42
VT	00001011	0B	11	+	00101011	2B	43
FF	00001100	0C	12	,	00101100	2C	44
CR	00001101	0D	13	-	00101101	2D	45
SO	00001110	0E	14	.	00101110	2E	46
SI	00001111	0F	15	/	00101111	2F	47
DLE	00010000	10	16	0	00110000	30	48
DC1	00010001	11	17	1	00110001	31	49
DC2	00010010	12	18	2	00110010	32	50
DC3	00010011	13	19	3	00110011	33	51
DC4	00010100	14	20	4	00110100	34	52
NAK	00010101	15	21	5	00110101	35	53
SYN	00010110	16	22	6	00110110	36	54
ETB	00010111	17	23	7	00110111	37	55
CAN	00011000	18	24	8	00111000	38	56
EM	00011001	19	25	9	00111001	39	57
SUB	00011010	1A	26	:	00111010	3A	58
ESC	00011011	1B	27	;	00111011	3B	59
FS	00011100	1C	28	<	00111100	3C	60
GS	00011101	1D	29	=	00111101	3D	61
RS	00011110	1E	30	>	00111110	3E	62
US	00011111	1F	31	?	00111111	3F	63

Codice ASCII a 8 bit (2/2)

carattere	codice ASCII	equivalente esadecimale	equivalente numerico	carattere	codice ASCII	equivalente esadecimale	equivalente numerico
@	01000000	40	64	‘	01100000	60	96
A	01000001	41	65	a	01100001	61	97
B	01000010	42	66	b	01100010	62	98
C	01000011	43	67	c	01100011	63	99
D	01000100	44	68	d	01100100	64	100
E	01000101	45	69	e	01100101	65	101
F	01000110	46	70	f	01100110	66	102
G	01000111	47	71	g	01100111	67	103
H	01001000	48	72	h	01101000	68	104
I	01001001	49	73	i	01101001	69	105
J	01001010	4A	74	j	01101010	6A	106
K	01001011	4B	75	k	01101011	6B	107
L	01001100	4C	76	l	01101100	6C	108
M	01001101	4D	77	m	01101101	6D	109
N	01001110	4E	78	n	01101110	6E	110
O	01001111	4F	79	o	01101111	6F	111
P	01010000	50	80	p	01110000	70	112
Q	01010001	51	81	q	01110001	71	113
R	01010010	52	82	r	01110010	72	114
S	01010011	53	83	s	01110011	73	115
T	01010100	54	84	t	01110100	74	116
U	01010101	15	85	u	01110101	75	117
V	01010110	56	86	v	01110110	76	118
W	01010111	57	87	w	01110111	77	119
X	01011000	58	88	x	01111000	78	120
Y	01011001	59	89	y	01111001	79	121
Z	01011010	5A	90	z	01111010	7A	122
[	01011011	5B	91	{	01111011	7B	123
\	01011100	5C	92	|	01111100	7C	124
]	01011101	5D	93	}	01111101	7D	125
^	01011110	5E	94	~	01111110	7E	126
_	01011111	5F	95	□	01111111	7F	127

Codice UNICODE

- I codici ASCII e EBCDIC progettati fondamentalmente in riferimento alla lingua inglese.
- Carenti per una utilizzazione nel rispetto delle molte lingue esistenti (v. ideogrammi della scrittura cinese, giapponese,..)
- La codifica UNICODE nasce per rappresentare ed unificare gli alfabeti di tutti i differenti linguaggi del pianeta.
  - che utilizza l'UTF (Unicode Transformation Format).
  - I formati UTF possono essere a 8, 16 e 32 bit.
- Utilizza una codifica a UTF-16 bit (65535 rappresentazioni possibili)
  - È la rappresentazione interna usata da Windows e Mac OS-X.
- Nello standard versione 3.0 ne codifica 49194
- I primi 256 sono quelli della codifica ASCII

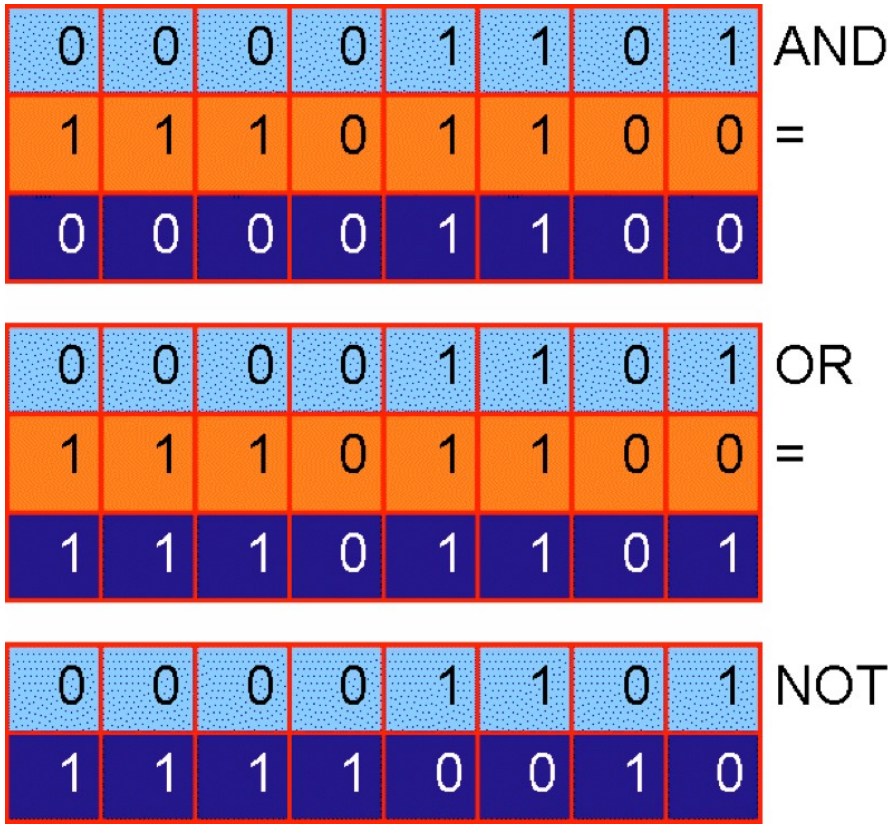
Operatori Booleani

- Sulle stringhe di bit sono anche definiti operatori che lavorano bit a bit (*bitwise operator*). Essi sono detti *booleani* e sono:
  - ***AND***: dati due bit restituisce il valore 1 se e solo se i bit erano entrambi posti a 1, in tutti gli altri casi il risultato è 0; l'AND è detto anche *prodotto logico*.
  - ***OR***: dati due bit restituisce il valore 0 se e solo se i bit erano entrambi posti a 0, in tutti gli altri casi il risultato è 1; l'OR è anche detto *somma logica*.
  - ***NOT***: dato un bit restituisce il valore 0 se esso era posto a 1, restituisce invece 1 se il bit era posto a 0; il NOT viene anche detto operatore di *negazione* o di *complementazione*.

Tavola di Verità

a	b	*NOT* a	a *AND* b	a *OR* b
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

Esempio


0 0 0 0 1 1 0 1
AND
1 1 1 0 1 1 0 0
=
0 0 0 0 1 1 0 0
0 0 0 0 1 1 0 1
OR
1 1 1 0 1 1 0 0
=
1 1 1 0 1 1 0 1
0 0 0 0 1 1 0 1
NOT
1 1 1 1 0 0 1 0

Sistemi Periodici

- Nel mondo reale esistono esempi di sistemi di numerazione basati su numeri a precisione finita come i sistemi di misura dei gradi degli angoli ([0, 360]) e del tempo ([0, 60] per minuti e secondi e [0, 24] per le ore).
- In tali sistemi si introduce il concetto di *periodicità*, per cui valori non compresi nell'intervallo di definizione vengono fatti ricadere in esso.
- Per la periodicità i valori esterni all'intervallo di definizione vengono ricondotti ad esso prendendo il resto della divisione dei valori per il periodo

intervallo	*periodo*	*Valore*	*divisione*	*resto*
[0,360]	360	1200	1200 : 360	120
[0,60]	60	61	61 : 60	1
[0,60]	60	55	55 : 60	55

Esercizi Riepilogativi

- Convertire in binario i seguenti numeri
  - 3,45
  - 40,08
  - 101,55
- Convertire in binario su 8 bit i seguenti numeri e valutarne l'approssimazione:
  - 0,007
  - 0,0012
  - 0,99
- Esprimere il numero binario 1101,101 in notazione posizionale e convertirlo in base 10.

Esercizi Riepilogativi

- Si dia la rappresentazione in complemento a 2 con parole di 8 bit dei seguenti numeri:
  - -123
  - 112
  - -100
- Che numero rappresenta la sequenza 11100110 di 8 bit se il sistema di rappresentazione è in complemento a 2?
- Che rappresentazione ha la sequenza di 8 bit in complemento a 2 11100110 se la parola viene allungata a 12 bit? E la sequenza 00010011?
- Si rappresenti il numero 123 in *eccesso-100* con parole di 8 bit.
- Calcolare la seguenti somme tenendo conto che gli addendi sono numeri espressi in complementi a 2
  - 1000 + 0110
  - 0111 + 0001
  - 1110 + 1001

Esercizi Riepilogativi

- Convertire il numero -32,475 in formato a virgola mobile IEEE 754 (precisione singola)
- Che numero rappresenta la seguente configurazione in IEEE 754 a singola precisione?

  0 10001100 10001100000000000000000

- Utilizzando la tabella indicare quali caratteri sono rappresentati dai seguenti codici ASCII a 7 bit:
  - 011 0110
  - 110 0101
  - 111 1111