Corso di

Architettura degli Elaboratori – B

A.A. 2018/2019

Codifica dell'informazione: Numeri a Virgola Mobile

Numeri in virgola mobile

In un calcolo *astronomico* è necessario esprimere:

- la massa dell'elettrone: 9×10^{-28} grammi
  0.0000000000000000000000000009
- la massa del sole: 2×10^{33} grammi
  2000000000000000000000000000000000

Quante cifre occorre usare?

62 cifre: 34 alla sinistra della virgola e 28 a destra

0000000000000000000000000000000000.0000000000000000000000000009
2000000000000000000000000000000000.0000000000000000000000000000

Problema: anche se la gamma dei numeri necessari è molto grande, i numeri significativi sono pochi……

Soluzione: notazione scientifica

Numeri in virgola mobile

- La notazione scientifica è un tipo di rappresentazione in cui la “**gamma**”dei numeri esprimibili è indipendente dal numero delle cifre significative.

  - **$n = m \times 10^{e}$**

mantissa o frazione

esponente

- la versione informatica di questa notazione è la rappresentazione in **virgola mobile** o **floating point**

Numeri in virgola mobile

Esempi:

3,14	$= 0{,}314 \times 10^{1}$	$= 3{,}14 \times 10^{0}$	$= 314 \times 10^{-2}$
- 0,0000005	$= -5 \times 10^{-7}$	$= -0{,}5 \times 10^{-6}$	
127000000	$= 127 \times 10^{6}$	$= 1{,}27 \times 10^{8}$	

La rappresentazione non è unica; esistono convenzioni che permettono di ottenere una rappresentazione unica, ad es. imponendo che la prima cifra significativa della mantissa si trovi immediatamente a destra della virgola; queste forme si dicono ***rappresentazioni normalizzate:***

3,14	$= 0{,}314 \times 10^{1}$
- 0,0000005	$= -0{,}5 \times 10^{-6}$
127000000	$= 0{,}127 \times 10^{9}$

Numeri in virgola mobile

Pensando ad una utilizzazione per il calcolatore si possono stabilire ulteriori convenzioni:

- fissare la lunghezza della mantissa ad un valore costante
- limitare l’esponente ad opportuni intervalli
- utilizzare un esponente convenzionale che lo renda sempre positivo (notazione in eccesso)
- disporre i tre elementi: <segno, esponente, mantissa> in un ordine stabilito

segno	*esponente*	*mantissa*

Numeri in virgola mobile

Esempio

- lunghezza mantissa: 8 cifre
- valore effettivo esponente: da -50 a +49
- in notazione eccesso 50 l'esponente e' sempre positivo (0 - 99)

i numeri – 0,0000005 e 127000000 si scrivono nel seguente modo:

	segno	*esponente*	*mantissa*
$- 0,5 \times 10^{-6}$	**-**	**44**	**50000000**
$0,127 \times 10^{9}$	**+**	**59**	**12700000**

Numeri in virgola mobile

Esempio di rappresentazione floating point binaria su 32 bit

- il primo bit rappresenta il segno della mantissa (0 per +, 1 per -)
- 7 bit successivi rappresentano l'esponente (espresso in base 2) in notazione eccesso 64 (esponente effettivo tra -64 e +63)
- gli ultimi 24 bit rappresentano la mantissa normalizzata

Spostando la virgola a sinistra (dividere per la base) si aumenta di 1 l'esponente (si moltiplica per la base) mantenendo l'uguaglianza

$\times 2^0$

$\times 2^8$

Esempio: **204,17437**

rappresentazione binaria: **11001100,00101100100001110**

rappresentazione normalizzata $0{,}110011000010110010000111 \times 10^{1000}$

- bit di segno: 0
- esponente eccesso 64: 1001000
- mantissa: 110011000010110010000111

0	1001000	110011000010110010000111

Numeri in virgola mobile

- La gamma (range) è determinata dal numero di cifre dell’esponente e la precisione dal numero di cifre della mantissa

ATTENZIONE!

- Con i numeri floating-point si può “simulare” il sistema dei numeri reali, pur con grandi differenze:
  - i numeri reali hanno la potenza del continuo
  - i numeri floating point sono in numero finito

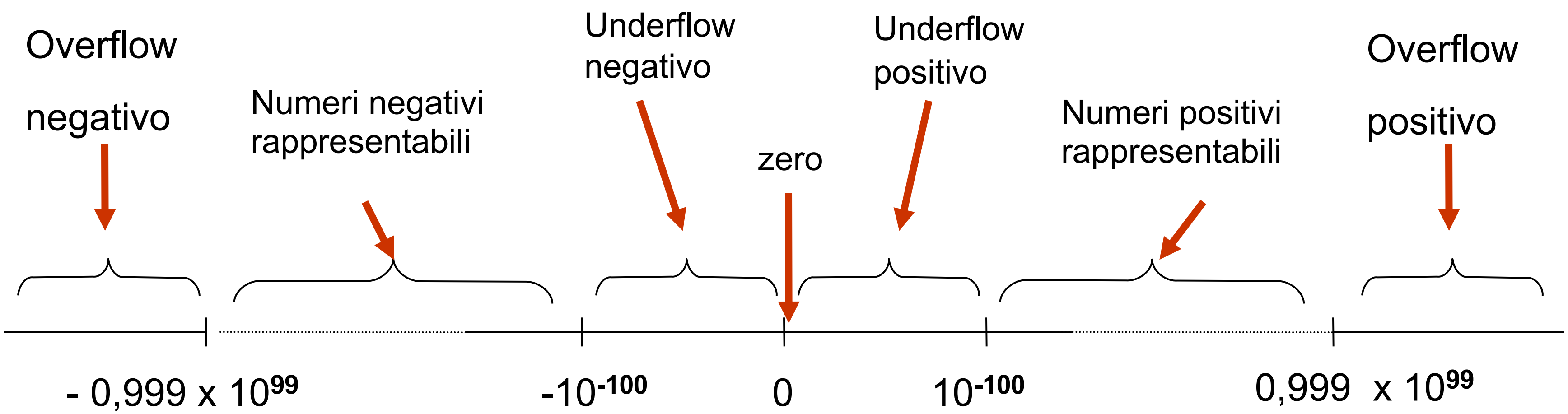
Numeri in virgola mobile

- Per esempio, consideriamo rappresentazioni (espresse in base 10) con una mantissa di tre cifre con segno nella gamma **$0,1 \le |m| < 1$** piu zero ed esponente di due cifre (con segno)

  - minimo numero negativo: -0,999 x 1099
  - massimo numero negativo: -0,100 x 10-99
  - minimo numero positivo: +0,100 x 10-99
  - massimo numero positivo: +0,999 x 1099

- Si rappresentano un numero finito di numeri negativi e numeri positivi, oltre allo zero, che ha tante rappresentazioni.

Numeri in virgola mobile



- “Spazio” tra numeri adiacenti non costante
- Arrotondamento
- Il numero di cifre della mantissa determina la densità dei punti, cioè la *precisione* delle approssimazioni
- Il numero di cifre dell’esponente determina la *dimensione degli intervalli* dei numeri rappresentabili

Standard IEEE 754

- Ogni produttore aveva un suo formato floating-point
- Fine anni '70 la IEEE costituisce un comitato al fine di standardizzare l'aritmetica floating-point
- Tre formati: singola precisione (32 bit), doppia precisione (64 bit), precisione estesa (80 bit)
- Base 2 per mantissa, notazione in eccesso per esponente
- Mantissa **normalizzata**: la parte intera è sempre 1, questo bit è nascosto quindi la mantissa si compone della sola parte frazionaria

Standard IEEE 754

Semplice precisione: 32 bit

1	8	23
	(-126;127)	mantissa

segno

esponente
eccesso 127

Min numero rappresentabile > 0 : 2^{-126} (10^{-38})

Max numero rappresentabile: appr. 2^{128} (10^{38})

Doppia precisione: 64 bit

1	11	52
	(-1022;1023)	mantissa

segno

esponente
eccesso 1023

Min numero rappresentabile > 0: 2^{-1022} (10^{-308})

Max numero rappresentabile: appr. 2^{1024} (10^{308})

Standard IEEE 754

432	0
216	0
108	0
54	0
27	1
13	1
6	0
3	1
1	1

$$1\,1\,0\,1\,1\,0\,0\,0\,0 = \cancel{1}.\,1\,0\,1\,1 \times 2^{8}$$

$$1\,1\,0\,1\,1\,0\,0\,0\,0 \times 2^{0} = 1\,.\,1\,0\,1\,1 \times 2^{8}$$

$$127+8 = 135|_{10} =$$

$$10000111|_{2}$$

8 bit

23 bit

0	1	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

segno

Esponente eccesso 127

mantissa

$23+8+1$

Standard IEEE 754

- Numeri normalizzati e denormalizzati
- Formati speciali per identificare infinito e NaN (Not a Number, esempio se dividiamo infinito per infinito)

	esp	*mantissa M*	*valore v*
Numero normalizzato	$0 < esp < 255$	qualunque	$v = (-1)^s(1,M)2^{esp-127}$
Numero denormalizzato	$esp = 0$	$M \neq 0$	$v = (-1)^s(0,M)2^{-126}$
Zero	$esp = 0$	$M = 0$	$v = (-1)^s0$
Infinito	$esp = 255$	$M = 0$	$v = (-1)^s\infty$
NaN	$esp = 255$	$M \neq 0$	$v =$ NaN

Codifica dei caratteri

- L'insieme di simboli comunemente usati nell'alfabeto anglosassone, incluse le cifre numeriche, lettere maiuscole e minuscole, simboli di punteggiatura, parentesi e operatori aritmetici, può essere codificato usando 7 bit ($2^7 = 128$)
- Codice **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)
- **UNICODE** su 16 bit: 65536 possibili caratteri
- **UTF-8** di lunghezza variabile da 1 a più byte

Il codice ASCII: alcuni caratteri

ASCII	Simb.	ASCII	Simb.	ASCII	Simb.
00101010	*	00111001	9	01000111	G
00101011	+	00111010	:	01001000	H
00101100	,	00111011	;	01001001	I
00101101	-	00111100	<	01001010	J
00101110	.	00111101	=	01001011	K
00101111	/	00111110	>	01001100	L
00110000	0	00111111	?	01001101	M
00110001	1	01000000	@	01001110	N
00110010	2	01000001	A	01001111	O
00110011	3	01000010	B	01010000	P
00110100	4	01000011	C	01010001	Q
00110101	5	01000100	D	01010010	R
00110110	6	01000101	E	01010011	S
00111000	8	01000110	F	01010100	T

Il codice ASCII

- Sebbene 7 bit siano sufficienti per codificare l'insieme di caratteri di uso comune, il codice ASCII standard utilizza 8 bit, il primo (più a sinistra) dei quali è sempre 0
- Esempio: codifica della parola **cane**
- **01100011 01100001 01101110 01100101**
  c a n e
- **Problema inverso: quale testo è codificato da una data sequenza?**
  - **si divide la sequenza in gruppi di otto bit (un byte);**
  - si determina il carattere corrispondente ad ogni byte

0110100101101100001000000101000001101111001101110

Il codice ASCII

- Tra i simboli speciali del codice ASCII vi è anche il simbolo spazio bianco “NUL”(codice 00000000), il simbolo di fine riga “CR” (00001101)
- In questo modo è possibile rappresentare mediante una sequenza di codici ASCII un testo strutturato in righe e pagine

Binario	Oct	Dec	Hex	Glifo	Binario	Oct	Dec	Hex	Glifo	Binario	Oct	Dec	Hex	Glifo
010 0000	040	32	20	Spazio	100 0000	100	64	40	@	110 0000	140	96	60	`
010 0001	041	33	21	!	100 0001	101	65	41	A	110 0001	141	97	61	a
010 0010	042	34	22	"	100 0010	102	66	42	B	110 0010	142	98	62	b
010 0011	043	35	23	#	100 0011	103	67	43	C	110 0011	143	99	63	c
010 0100	044	36	24	$	100 0100	104	68	44	D	110 0100	144	100	64	d
010 0101	045	37	25	%	100 0101	105	69	45	E	110 0101	145	101	65	e
010 0110	046	38	26	&	100 0110	106	70	46	F	110 0110	146	102	66	f
010 0111	047	39	27	'	100 0111	107	71	47	G	110 0111	147	103	67	g
010 1000	050	40	28	(	100 1000	110	72	48	H	110 1000	150	104	68	h
010 1001	051	41	29	)	100 1001	111	73	49	I	110 1001	151	105	69	i
010 1010	052	42	2A	*	100 1010	112	74	4A	J	110 1010	152	106	6A	j
010 1011	053	43	2B	+	100 1011	113	75	4B	K	110 1011	153	107	6B	k
010 1100	054	44	2C	,	100 1100	114	76	4C	L	110 1100	154	108	6C	l
010 1101	055	45	2D	-	100 1101	115	77	4D	M	110 1101	155	109	6D	m
010 1110	056	46	2E	.	100 1110	116	78	4E	N	110 1110	156	110	6E	n
010 1111	057	47	2F	/	100 1111	117	79	4F	O	110 1111	157	111	6F	o
011 0000	060	48	30	0	101 0000	120	80	50	P	111 0000	160	112	70	p
011 0001	061	49	31	1	101 0001	121	81	51	Q	111 0001	161	113	71	q
011 0010	062	50	32	2	101 0010	122	82	52	R	111 0010	162	114	72	r
011 0011	063	51	33	3	101 0011	123	83	53	S	111 0011	163	115	73	s
011 0100	064	52	34	4	101 0100	124	84	54	T	111 0100	164	116	74	t
011 0101	065	53	35	5	101 0101	125	85	55	U	111 0101	165	117	75	u
011 0110	066	54	36	6	101 0110	126	86	56	V	111 0110	166	118	76	v
011 0111	067	55	37	7	101 0111	127	87	57	W	111 0111	167	119	77	w
011 1000	070	56	38	8	101 1000	130	88	58	X	111 1000	170	120	78	x
011 1001	071	57	39	9	101 1001	131	89	59	Y	111 1001	171	121	79	y
011 1010	072	58	3A	:	101 1010	132	90	5A	Z	111 1010	172	122	7A	z
011 1011	073	59	3B	;	101 1011	133	91	5B	[	111 1011	173	123	7B	{
011 1100	074	60	3C	<	101 1100	134	92	5C	\	111 1100	174	124	7C	|
011 1101	075	61	3D	=	101 1101	135	93	5D	]	111 1101	175	125	7D	}
011 1110	076	62	3E	>	101 1110	136	94	5E	^	111 1110	176	126	7E	~
011 1111	077	63	3F	?	101 1111	137	95	5F	_					

Numeri: ASCII vs binario

- I numeri possono essere codificati in due modi:
  - <u>ASCII</u>: **37** **00110011 00110111** (2 byte)
  - <u>BINARIO</u>: **37** **00100101** (1 byte)
- Il primo modo è usato per le comunicazioni con l'esterno (input/output)
- Il secondo modo è usato all'interno; per fare i calcoli non è possibile usare direttamente le codifiche ASCII (Esistono dei programmi di conversione che trasformano i numeri da una codifica all'altra)
- **Esempio**

  3 + 2 = e (vi ricordate? Una codifica dove la somma delle rappresentazioni restituisce la rappresentazione della somma)

  (in ASCII: **00110011 + 00110010 = 01100101**)

La codifica UNICODE

- 52 lettere alfabetiche maiuscole e minuscole, 10 caratteri che denotano le cifre (0, 1, 2, ..., 9), segni di punteggiatura (, . ; : ! ” ? ’ ^ \ ...), simboli matematici (+, -, × , ±, {, [, >, ...), caratteri di alfabeti nazionali (à, è, ì, ò, ù, ç, ñ, ö, ...), altri segni grafici (©, ←, ↑, □, @, €, ...)
- 16 bit per carattere: la codifica di quelli presenti in ASCII è la stessa sugli 8 bit meno significativi (gli 8 più significativi sono a 0). Ad esempio: il carattere **K** è codificato in:
  - ASCII come **01001011**
  - UNICODE come **0000000 01001011**
- Il carattere **é** invece (non presente in ASCII) è codificato come **0000000 11101001**
- Le lingue di tutto il mondo utilizzano all’incirca 200000 simboli: UNICODE non sufficiente

La codifica UTF-8

- Lunghezza variabile da 1 a più byte
- I caratteri del codice ASCII sono codificati con un byte con gli stessi valori (UNICODE usa 2 byte!!)
- Se il primo byte ha il bit più significativo a 1 (in ASCII è sempre 0!!) allora la codifica si estende su uno o più byte successivi (i cui due primi bit sono sempre uguali a 10)
- Il valore dei primi bit determina il numero di byte successivi secondo il seguente schema

La codifica UTF-8

Bits	Bytes	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	1	0xxxxxxx					
11	2	110xxxxx	10xxxxxx				
16	3	1110xxxx	10xxxxxx	10xxxxxx			
21	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

Ad esempio:

- il carattere **K** è codificato in:
  - ASCII come **01001011**
  - UNICODE come **0000000 01001011**
  - UTF-8 come **01001011**
- il carattere é è codificato in
  - ASCII (non codificato!!)
  - UNICODE come **0000000 11001001**
  - UTF-8 come **11000011 10101001**

Qualche problema

- Problema 1: Ipotizziamo l'esistenza del formato IEEE 754 in precisione ***scarsa*** su 8 bit con 1 bit di segno, 4 bit di esponente e 3 bit di mantissa (nell'ordine)**.** Esso condivide tutte le caratteristiche dei formati a precisione singola e doppia dello standard IEEE 754, ovvero, la codifica dei numeri normalizzati, denormalizzati, dello zero, dell'infinito, del NaN, con le stesse convenzioni di codifica. Dire:
  - In eccesso a quale valore sarebbe codificato l'esponente?
  - Che numero sarebbe rappresentato dalla sequenza 11011100?
- Problema 2: Descrivere la notazione standard IEEE 754 in semplice precisione, specificando i campi che la compongono e descrivendo i passi da eseguire per codificare il numero 13,75.
- Problema 3: Decodificare la sequenza
  1 10000011 11100000000000000000000
  assumendo che rappresenti un numero in formato IEEE 754 in precisione singola