Esempi di rappresentazione nello standard IEEE 754

- Convertire il numero **-5.828125_{10}** in formato a virgola mobile IEEE 754 (precisione singola) e provare la correttezza della conversione tramite procedimento inverso

Esempi di rappresentazione nello standard IEEE 754

- Convertire il numero **-5.828125_{10}** in formato a virgola mobile IEEE 754 (precisione singola) e provare la correttezza della conversione tramite procedimento inverso

5 → 101

.828125 → 110101

101.110101 == $1.01110101 * 2^{+2}$

Segno: - → 1

Mantissa: 01110101

Esponente: 2+127 = 129 = 10000001

P754: 1 10000001 01110101000000000000000

Esempi di rappresentazione nello standard IEEE 754

- Convertire il numero **-5.828125_{10}** in formato a virgola mobile IEEE 754 (precisione singola) e provare la correttezza della conversione tramite procedimento inverso

  Segno: 1 → -

  Mantissa: 01110101 = $(2^{-2}+2^{-3}+2^{-4}+2^{-6}+2^{-8})+1$ = 1.45703125

  Esponente: 10000001 = 129-127 = +2

  P754: $- 1.45703125 \times 2^{+2} = -5.828125$

Numeri denormalizzati

- Per i numeri denormalizzati non vale la regola del significando: hanno uno zero a sinistra della virgola.
- Il numero denormalizzato più grande ha esponente 0 e tutti uni nella mantissa, e vale circa $0.9999999 \times 2^{-126}$, quindi quasi come il più piccolo normalizzato.
  - 0.9999999 è dato dalla mantissa, il fattore 2^{-126} è convenzionale.
- È però possibile comporre numeri denormalizzati progressivamente più piccoli azzerando i bit a sinistra della mantissa.
- Il più piccolo numero denormalizzato ha la mantissa composta di zeri, con il solo LSB pari a uno (0000...001). Il suo valore è 2^{-149}.
  - 2^{-23} dato dalla mantissa e 2^{-126} convenzionale
- In pratica i numeri denormalizzati permettono una “transizione morbida” verso i valori che causano l’underflow.

Gli altri formati

- Lo zero è rappresentato esplicitamente da una configurazione apposita.
- Si può rappresentare il valore “infinito”
  - Che consente operazioni del tipo infinito + costante = infinito
- Se si fanno operazioni del tipo infinito / infinito si ottiene la codifica di qualcosa che non è un numero (not a number).

Operazioni in virgola mobile

- Somma e sottrazione:
  - si uguagliano gli esponenti
  - le mantisse vengono sommate
  - si rinormalizza la mantissa (con aggiustamento dell'esponente)

- Moltiplicazione e divisione:
  - si moltiplicano o si dividono le mantisse in modo consueto
  - si sommano o si sottraggono gli esponenti
  - si normalizza (si riporta in forma normale)

Operazioni in virgola mobile

- Operazioni in virgola mobile
  - assai più complesse delle op sui numeri interi (con o senza segno)
  - spesso anche più ottimizzate
  - in molti contesti, sono le op più utili e comuni dell'elaborazione

- Potenza di calcolo matematica: spesso espressa in **FLOPS**
  FLoating-pont **OP**eration per **S**econd (operazioni in virgola mobile al secondo)
  - K-FLOPS (kilo-flops) : migliaia di op al sec (anni 50)
  - M-FLOPS (mega-flops): milioni di op al sec (anni 60-70)
  - G-FLOPS (giga-flops): miliardi di op al sec (anni 80-90)
  - T-FLOPS (tera-flops): migliaia di miliardi di op al sec (anni 2000)
  - P-FLOPS (peta-flops): milioni di miliardi di op al sec (anni 2010)

I numeri reali nei linguaggi di programmazione

- In molti linguaggi (C, Java…)
  si possono dichiarare variabili in singola o doppia precisione
- Singola precisione (tipicamente FLOAT-32): `float x;`
- Doppia precisione (tipicamente FLOAT-64): `double x;`
- I letterali frazionari sono double per default:
  `0.5` → literal in doppia precisione.
- Per indicare che un letterale è a singola precisione (float) si appone una f:
  `0.5f` → literal in precisione singola.

Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

La rappresentazione dei caratteri

I caratteri

- Il carattere è l'elemento base delle lingue umane.
- Problema: in alfabeti diversi i caratteri hanno particolarità diverse:
  - Negli alfabeti di derivazione greca (greco, latino e cirillico), esiste la distinzione tra maiuscole e minuscole, ignota altrove
  - Negli alfabeti di derivazione latina si sono inventati segni particolari sulle lettere, come ad es. gli accenti.
  - In ebraico, le vocali sono modificatori di lettere di consonanti
  - In arabo, la giustapposizione di lettere diverse nella parola provoca una differenziazione della forma delle lettere stesse.
  - In cinese, è possibile creare nuovi caratteri come composizione di altri caratteri esistenti.

La codifica dei caratteri

- Iniziamo a considerare i caratteri base degli alfabeti occidentali, che, per motivi storici, sono stati i primi a essere codificati.
- Prima osservazione: i caratteri sono relativamente pochi
  - 10 cifre numeriche
  - 26 minuscole
  - 26 maiuscole
  - una trentina di simboli di interpunzione (,?!;:…) e altri simboli di uso comune (@#$|/\)
  - In totale un centinaio di caratteri: un byte (256 combinazioni) ci basta e avanza per rappresentarli tutti.
- Seconda osservazione: mentre per i numeri esiste una codifica “naturale” (ad es. lo zero è rappresentato da un insieme di bit nulli) per i caratteri no (non fa alcuna differenza se la lettera “r” è codificata come 35 o 77).

La codifica dei caratteri

- Il problema della rappresentazione è quindi semplice: basta fissare una corrispondenza tra numeri appartenenti all'intervallo 0-255 e i caratteri.
- La cosa importante è che questa corrispondenza sia nota ed accettata da tutti (pena la scorretta interpretazione dei testi).
- In conclusione, la codifica dei caratteri deve essere stabilita da uno standard.

Gli standard

- Lo standard più noto e quasi universalmente adottato è l'ASCII (American Standard Code for Information Interchange). È uno standard ANSI (X3.4 - 1968) che definisce valori per 128 caratteri, di cui 33 (0-31 e 127) non stampabili.
- EBCDIC (Extended Binary Characters for Digital Interchange Code) è il codice caratteri a 8 bit usato da IBM nei suoi mainframe.

La tabella ASCII

	0	1	2	3	4	5	6	7
0			SPC	0	@	P	'	p
1			!	1	A	Q	a	q
2			"	2	B	R	b	r
3			#	3	C	S	c	s
4	EOT		$	4	D	T	d	t
5			%	5	E	U	e	u
6			&	6	F	V	f	v
7			'	7	G	W	g	w
8	BSP		(	8	H	X	h	x
9			)	9	I	Y	i	y
A	LF		*	:	J	Z	j	z
B			*	;	K	[	k	{
C	FF		,	<	L	\	l	|
D	CR		-	=	M	]	m	}
E			.	>	M	^	n	~
F			/	?	O	_	o	del

I caratteri “di controllo”

- Per gestire un testo non servono solo i caratteri visibili (stampabili), ma ci vogliono anche un certo numero di caratteri “di controllo”.
  - Ad es. per segnalare la fine di una riga sono disponibili CR (carriage return) e LF (line feed)
  - EOT (end of text) indica la fine del testo
  - Del indica la cancellazione di un carattere precedentemente inserito
- Tutti questi caratteri non sono visualizzati, hanno invece un effetto sul testo.

ISO 8859/1 (ISO Latin-1)

- Il codice ASCII sfrutta solo 7 degli 8 bit di un byte, e non è pienamente soddisfacente (ad es. gli mancano tutte le lettere accentate).
- L'ISO Latin-1 –molto utilizzato nelle pagine WEB– estende il codice ASCII nel senso che usa tutti gli 8 bit di un byte, rappresentando quindi 256 caratteri diversi.
  - Oltre ai caratteri ASCII, ISO Latin-1 contiene vari caratteri accentati e lettere usate dai linguaggi dell'Europa occidentale (Italiano, Francese, Spagnolo, Tedesco, Danese, ecc.): ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿ À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ð Ñ 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 ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ
- L'ISO Latin-1 è compatibile con l'ASCII nel senso che i primi 128 codici hanno identico significato nei due standard.

La gestione degli altri alfabeti

- Se vogliamo scrivere un testo in caratteri greci o cirillici come facciamo?
- Con ASCII o ISO Latin-1 non è possibile.
- Sono stati inventati diversi codici a 8 bit per alfabeti non latini (e.g., cirillico, greco e giapponese semplificato) e molti codici a 16 bit per linguaggi orientali (giapponese e cinese).
- Problemi:
  - Un programma che “capisce” un certo codice non sarà capace di trattare codici diversi.
  - Non possiamo usare nello stesso testo caratteri cirillici e greci
- UNICODE e ISO/IEC 10646 sono standard che consentono di rappresentare tutti gli insiemi di caratteri noti, utilizzando un numero variabile di byte.

Interpretazione delle informazioni

- In un byte della memoria c'è la configurazione 01000001.
- Come fa il calcolatore a sapere se questo byte va interpretato come un numero intero (65) o come un carattere (A)?
- La risposta è semplice: non lo sa per niente.
- Sono i nostri programmi che devono ricordarsi qual è il significato di una data cella di memoria (o variabile).
- Molti linguaggi facilitano questo compito assegnando un "tipo" alle variabili, per cui su una variabile di tipo carattere non si possono fare somme.
- Altri linguaggi come gli assembler non danno questo tipo di protezione