UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

Dispense del corso di Fondamenti di Informatica 1

Rappresentazione dell'informazione

Ultimo aggiornamento: 19/09/2020

Informazione

- L'informazione (limitandoci al suo significato tecnico) è una sequenza di simboli che può essere interpretata come un messaggio (un trasferimento di conoscenza).
- L'informazione è qualsiasi tipo di evento che influenza lo stato di un sistema dinamico, in grado quindi di interpretarlo.
- Concettualmente, l'informazione è il messaggio che viene trasmesso. Quindi in generale l'informazione è «conoscenza comunicata o ricevuta relativa ad un particolare fatto o circostanza», o anche la risposta ad una domanda.
- L'informazione non può essere predetta e quindi risolve una incertezza. (http://en.wikipedia.org/wiki/Information)
- La parola deriva dal latino: *informatio*, ovvero «dare forma (alla mente/alle idee)».
- Quando «parliamo» trasferiamo qualche tipo di conoscenza all'ascoltatore. Quando «vediamo» acquisiamo informazioni sul mondo e sull'ambiente in cui siamo, o sul contenuto di un immagine o di un video.
- Tutti i nostri sensi ci forniscono informazioni.

Rappresentare l'informazione

- Gli uomini si sono ingegnati, a partire dalla preistoria, per trovare soluzioni in grado di trasmettere l'informazione.
- A partire dal linguaggio, si è passati ai disegni sulle pareti rocciose, ai segnali di fumo, fino ad arrivare alla scrittura e infine ai sistemi di comunicazione più moderni.
- Il calcolatore è in particolare una macchina in grado di elaborare informazioni, ma anche di memorizzarle, visualizzarle e trasmetterle.
- Prima di poter fare tutto questo, è necessario trovare quindi un modo di codificare le informazioni. In particolare il modo deve essere compatibile con le capacità del computer.
- Le informazioni a cui faremo riferimento sono:
  - numeri (quantità, misure, valori,…)
  - parole (testi)
  - immagini (statiche o in movimento)
  - suoni e musica
- In linea di principio si potrebbe voler rappresentare anche odori, sensazioni tattili o sapori.

Rappresentare i numeri

- L'informazione più elementare da elaborare con il calcolatore è il numero. In particolare cominciamo con i numeri interi non negativi.
- Si potrebbero utilizzare molte soluzioni diverse per rappresentare i numeri. I Romani ad esempio utilizzavano un sistema *additivo*. Una serie di segni convenzionali indicavano delle quantità notevoli e l'elenco di questi significava sommarne i valori.
- I segni base erano i seguenti:

I	V	X	L	C	D	M
1	5	10	50	100	500	1000

- Per convenzione si indicavano prima le quantità più grandi e poi quelle più piccole.
- Le regole però sono un po' più complicate, dato che se uno tra I,X,C viene preposto ad un simbolo di valore maggiore questo viene *sottratto* al numero successivo.

Sistema posizionale

- Successivamente si diffuse un sistema originario dell'India, che venne importato dagli Arabi e quindi giunse in Europa: il sistema *posizionale*.

- In questo sistema lo stesso simbolo cambia valore a seconda della posizione che assume nel numero.

- Elemento fondamentale è la *base* a cui si fa riferimento.

- Il valore di un numero è dato dalla somma dei valori dei simboli numerici (ognuno di valore inferiore alla base, quando preso singolarmente) moltiplicato per una appropriata potenza della base.

- Nell'uso comune utilizziamo la base 10 e 10 diversi simboli: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Sistema posizionale

- Consideriamo i numeri naturali, che chiameremo anche interi senza segno (*unsigned integer*):

$$x = \sum_{i=0}^{+\infty} \underset{b}{x_i} b^i = (\dots x_2 x_1 x_0)_b$$

- Indichiamo con $\underset{b}{x_i}$ l'i-esima cifra del numero x rappresentato in base b.

  Se raggruppiamo le cifre, possiamo indicare la base al pedice della parentesi.
- Normalmente, nei paesi europei, i numeri vengono scritti da sinistra a destra dalla cifra **non nulla** più significativa (di valore più alto) a quella meno significativa (di valore più basso).
- Ovvero il numero centotrentadue in base 10 si scrive $(132)_{10}$ e non $(\dots 00000000000132)_{10}$. Omettiamo infatti tutti gli infiniti zeri che corrispondono alle potenze di 10 superiori alla 2.

- L'aritmetica che utilizziamo tutti i giorni utilizza questa rappresentazione e la base che utilizziamo è la base 10.

Numeri naturali

- Scriviamo quindi

  $(168)_{10}$ per indicare $1 \times 10^2 + 6 \times 10^1 + 8 \times 10^0$.
- Ma anche nella descrizione che abbiamo scritto abbiamo utilizzato la base 10. Pertanto se in una sequenza numerica omettiamo l'indicazione della base sottintendiamo la base 10.
- Utilizzando un'altra base (ad esempio la base 4) dobbiamo scegliere un insieme di 4 simboli che corrispondano ai valori (valgano) 0,1,2,3. Ad esempio ♈=0, ♉=1, ♊=2, ♋=3. E potremmo scrivere:

  $$(♉♈♋)_4 = 1 \times 4^2 + 0 \times 4^1 + 3 \times 4^0 = 19$$
- Potevamo anche usare l'associazione ٠=0, ١=1, ٢=2, ٣=3, quindi:

  $$(١٠٣)_4 = 19$$
- Ovviamente tutto è possibile, ma in generale si preferisce utilizzare dei simboli già noti, per cui è più naturale in Italia utilizzare i simboli «0», «1», «2», «3». Quindi si scrive:

  $$(103)_4 = 1 \times 4^2 + 0 \times 4^1 + 3 \times 4^0 = 19$$
- Se non c'è rischio di confusione, possiamo omettere le parentesi attorno al numero e scrivere: $103_4 = 19_{10} = 19$

Numeri naturali

- Vediamo altri esempi:

$$641_7 = 6 \times 7^2 + 4 \times 7^1 + 1 \times 7^0 = 323$$

$$1255_6 = 1 * 6^3 + 2 * 6^2 + 5 * 6^1 + 5 * 6^0 = 323$$

- Quindi $641_7 = 1255_6$.
- La base più piccola che consente di utilizzare un sistema posizionale è la base 2. Il numero precedente diventerebbe:

$$101000011_2$$
$$= 1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 256 + 0 + 64 + 0 + 0 + 0 + 0 + 2 + 1 = 323$$

- La base 2 è molto importante, dato che è quella usata dai calcolatori elettronici.
- Ogni cifra binaria è detta *bit*. Un gruppo di 8 bit è detto *byte*.
- Per semplificarne la lettura le cifre binarie vengono scritte raggruppate 4 a 4 a partire da quelle meno significative, quindi il numero precedente viene scritto come: 1.0100.0011

Rappresentare un valore in base *b*

- Qual è la rappresentazione del valore 17 in base 5?

$$17 = (32)_5$$
$$17 = 3 \times 5^1 + 2 \times 5^0$$

- Qual è la rappresentazione del valore 53 in base 6?

$$53 = (125)_6$$
$$53 = 1 \times 6^2 + 2 \times 6^1 + 5 \times 6^0$$

- Qual è la rappresentazione del valore 138 in base 7?

$$138 = (255)_7$$
$$138 = 2 \times 7^2 + 5 \times 7^1 + 5 \times 7^0$$

- C'è un procedimento sistematico per ottenere queste conversioni? Una serie di passaggi ripetibili? Insomma esiste un *algoritmo*?

- Un **algoritmo** è un procedimento che risolve una classe di problemi attraverso un numero finito di istruzioni elementari, chiare e non ambigue.

Ottenere la rappresentazione in una base *b*

- Ricordate il concetto di divisione intera?
- Dividere il numero x per il divisore $d \neq 0$, significa trovare il quoziente q e il resto $r < d$ tali che:

$$x = q \times d + r$$

In che base è x?

In nessuna!

- Possiamo notare che utilizzando la rappresentazione posizionale,

$$x = \sum_{i=0}^{+\infty} \underset{b}{x_i} b^i = \sum_{i=1}^{+\infty} \underset{b}{x_i} b^i + \underset{b}{x_0} b^0 = \sum_{i=1}^{+\infty} \underset{b}{x_i} b^i + \underset{b}{x_0}$$

- E raccogliendo b nella sommatoria:

$$x = \left(\sum_{i=1}^{+\infty} \underset{b}{x_i} b^{i-1} \right) \times b + \underset{b}{x_0}$$

- Ovviamente possiamo aggiustare l'indice della sommatoria facendolo partire da 0:

$$x = \left(\sum_{i=0}^{+\infty} \underset{b}{x_{i+1}} b^i \right) \times b + \underset{b}{x_0}$$

Ottenere la rappresentazione in una base *b*

- Confrontando questo risultato con la definizione di divisione intera, abbiamo che se dividiamo un numero per b si ha che:

$$r = {}_{b}x_0$$

$$q = \sum_{i=0}^{+\infty} {}_{b}x_{i+1} b^i$$

- Quindi dividendo un numero non negativo per una base, il resto è la cifra meno significativa della sua rappresentazione in quella base.
- Ripetendo poi l'operazione sul quoziente così ottenuto, otterremmo la cifra successiva. Vediamo cosa accade lavorando in base 10:

137 : 10 = 13 con resto di 7

13 : 10 = 1 con resto di 3

1 : 10 = 0 con resto di 1

- Leggendo i resti dall'ultimo al primo osserviamo quindi tutte le cifre che compongono il valore 137.
- Nulla del ragionamento però dipende dalla base 10 e quindi possiamo applicare lo stesso metodo anche in altre basi.

Ottenere la rappresentazione in una base *b*

- Consideriamo nuovamente il valore 323 visto in precedenza e proviamo ad applicare il metodo visto ora con la base 7:

  323 : 7 = 46 con resto di 1

  46 : 7 = 6 con resto di 4

  6 : 7 = 0 con resto di 6

- Raccogliendo i resti dall'ultimo al primo: $323 = (641)_7$.
- Proviamo con la base 6:

  323 : 6 = 53 con resto di 5

  53 : 6 = 8 con resto di 5

  8 : 6 = 1 con resto di 2

  1 : 6 = 0 con resto di 1

- Quindi: $323 = (1255)_6$.
- Vediamo infine il procedimento in base 2.

Ottenere la rappresentazione in una base *b*

323	: 2 =	161	con resto di 1
161	: 2 =	80	con resto di 1
80	: 2 =	40	con resto di 0
40	: 2 =	20	con resto di 0
20	: 2 =	10	con resto di 0
10	: 2 =	5	con resto di 0
5	: 2 =	2	con resto di 1
2	: 2 =	1	con resto di 0
1	: 2 =	0	con resto di 1

- Quindi: $323 = (101000011)_2$.
- Il procedimento in base 2 è particolarmente semplice, dato che se il numero è pari il resto sarà zero, altrimenti sarà 1.
- Nel fare questo procedimento a mano è poi conveniente non ricopiare tutte le volte il risultato, scrivendolo direttamente nella riga sottostante.
- Infine possiamo anche omettere il « : 2 = ».

Ottenere la rappresentazione in base 2

- Vediamo quindi come possiamo scrivere l'esempio di prima:

323	1
161	1
80	0
40	0
20	0
10	0
5	1
2	0
1	1

- Notate che in pratica ci chiediamo ad ogni riga: «il numero è pari?». 1 vuol dire no, 0 vuol dire sì. Poi dividiamo per due (tenendo la parte intera).

Le basi 8 e 16

- Scrivere i numeri in base 2 costringe a utilizzare un numero elevato di cifre e non è sempre agevole.
- Per questo motivo si utilizzano due basi, potenze del due, che sono facilmente convertibili da e verso la base 2.
- Riveste un'importanza storica la base 8:

In base 10	In base 8	In base 2
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111

Le basi 8 e 16

- La rappresentazione in base 8 è detta *ottale* e consente di convertire il binario semplicemente per sostituzione con gruppi di tre cifre.
- Il valore $(111010101)_2 = (725)_8$, infatti $(111)_2 = (7)_8$, $(010)_2 = (2)_8$, $(101)_2 = (5)_8$.
- La corrispondenza è biunivoca e quindi la conversione è intuitiva.
- Il vero problema è che normalmente si utilizza un numero di cifre binarie potenza del due e in particolare 8, 16, 32, 64 sono i raggruppamenti più frequenti.
- Questi numeri non sono divisibili per 3, quindi non hanno una rappresentazione ottale in cui a ogni cifra corrispondono esattamente 3 bit: la cifra più significativa è sempre usata parzialmente.
- Gruppi di 4 bit sono invece più adatti, quindi si usa la rappresentazione in base 16, detta *esadecimale* (*hexdecimal* in inglese).
- Servono quindi 16 simboli diversi. Da 0 a 9 non ci sono problemi, ma per i valori da 10 a 15? Si è scelto di utilizzare le lettere da A a F.

Le basi 8 e 16

In base 10	In base 16	In base 2
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Le basi 8 e 16

- Alcuni esempi di conversioni:

$$(1101.0101)_2 = (D5)_{16}$$
$$(0111.1100.0000.1111)_2 = (7C0F)_{16}$$
$$(0110.1101.1110.0010.0101.0001.1010.1001)_2 = (6DE251A9)_{16}$$

- Anche un numero enorme a 32 cifre binarie, può essere rappresentato in modo molto compatto con 8 cifre esadecimali.
- Conoscendo bene la base 16 è possibile a mente vedere il valore dei singoli bit.
- Le conversioni da e per la base 10 possono essere eseguite utilizzando il metodo già visto per basi generiche.

I numeri con la virgola

- Quando vogliamo rappresentare quantità inferiori all'unità, possiamo ricorrere alla rappresentazione con la virgola.
- Anche questa è una rappresentazione posizionale, dove ogni posizione dopo la virgola è una **potenza negativa** della base:

$$x = \sum_{i=-\infty}^{+\infty} x_i b^i = (\dots x_2 x_1 x_0, x_{-1} x_{-2} \dots)_b$$

- È importante ricordare che non tutti i numeri razionali sono rappresentabili con un numero finito di cifre in questa forma, in tutte le basi. Ad esempio il valore $\frac{1}{3}$ non è rappresentabile con un numero finito di cifre in base 10: 0,3333... Invece il numero $\frac{1}{2}$ si rappresenta come 0,5.
- Lo stesso vale per le altre basi: $\frac{1}{3} = (0{,}1)_3$, mentre $\frac{1}{2} = (0{,}1111 \dots)_3$
- Vale la pena notare che siccome $10 = 2 \times 5$, qualsiasi numero frazionario rappresentabile con un numero finito di cifre in base 2 lo è anche in base 10, ma non è vero il viceversa.

Conversione di base

- Come possiamo effettuare la conversione dalla base 10 ad un'altra base con i numeri con la virgola? Notiamo che

$$x = \sum_{i=-\infty}^{+\infty} {}_{b}x_i b^i = \sum_{i=0}^{+\infty} {}_{b}x_i b^i + \sum_{i=-1}^{-\infty} {}_{b}x_i b^i$$

- Il primo termine sappiamo già convertirlo. Restano da estrarre i termini che moltiplicano le potenze negative, ma è chiaro che

$$\left(\sum_{i=-1}^{-\infty} {}_{b}x_i b^i\right) \times b = \sum_{i=-1}^{-\infty} {}_{b}x_i b^{i+1} = \sum_{i=0}^{-\infty} {}_{b}x_{i-1} b^i = {}_{b}x_{-1} + \sum_{i=-1}^{-\infty} {}_{b}x_{i-1} b^i$$

- ovvero che moltiplicando la parte frazionaria per una base otteniamo un numero la cui parte intera è la prima cifra dopo la virgola della rappresentazione in base b del numero (quella che moltiplicava b^{-1}).
- La parte frazionaria rimanente è costituita dalle cifre successive, spostate in avanti di una posizione. Quindi è possibile iterare il procedimento ed estrarre così tutte le cifre necessarie.
- Per quanto detto prima però, questo procedimento potrebbe non terminare mai!

Conversione di numeri con la virgola

- Consideriamo ad esempio 12,8125 e convertiamolo in base 4. La parte intera sarà $(30)_4$. Procediamo con la parte frazionaria:

0,8125 x 4 = 3,25 la cui parte intera è 3. Devo convertire 0,25

0,25 x 4 = 1 la cui parte intera è 1. Non resta nulla da convertire

- Quindi $12{,}8125 = (30{,}31)_4$. Facciamo lo stesso in base 5

0,8125 x 5 = 4,0625 la cui parte intera è 4. Devo convertire 0,0625

0,0625 x 5 = 0,3125 la cui parte intera è 0. Devo convertire 0,3125

0,3125 x 5 = 1,5625 la cui parte intera è 1. Devo convertire 0,5625

0,5625 x 5 = 2,8125 la cui parte intera è 2. Devo convertire **0,8125**

...

- Ho ottenuto un numero di quelli già convertiti precedentemente, pertanto da qui in poi si ripeterà all'infinito questa sequenza: il numero è periodico in base 5: $12{,}8125 = (22{,}\overline{4012})_5$.

I numeri con segno

- Quando vogliamo estendere la rappresentazione ai numeri con segno normalmente utilizziamo la *rappresentazione in modulo e segno*: un simbolo non numerico indica il verso sulla retta dei numeri (+ o -) e i numeri indicano con la rappresentazione posizionale la distanza dallo zero.

- Lo stesso procedimento può essere utilizzato rappresentando i numeri in una base arbitraria.

- Possiamo quindi scrivere $-12 = -1100_2 = -110_3 = -30_4 = -22_5 = -20_6 = -15_7 = -14_8 = -13_9 = -C_{16}$.

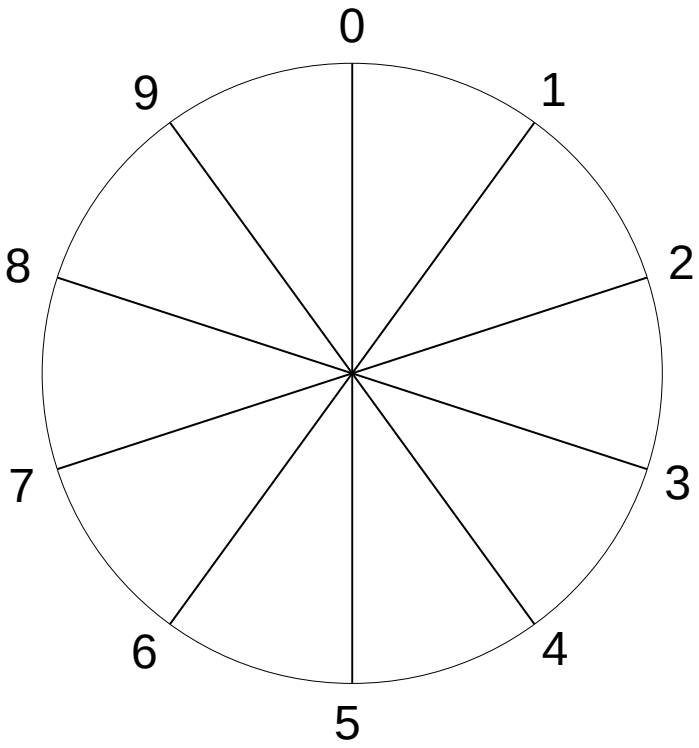
- Questa soluzione adottata normalmente non è la scelta più comoda nei calcolatori e vedremo soluzioni alternative.

Rappresentazione con un numero finito di cifre

- Normalmente, nella matematica tradizionale, non si impone alcun vincolo al massimo numero di cifre utilizzabile per rappresentare i numeri.
- In informatica però dobbiamo realizzare delle macchine che eseguano questi calcoli, quindi non è pensabile di lasciare illimitata la dimensione dei numeri.
- Vediamo ora che cosa accade quando il numero di cifre non può crescere a piacere, utilizzando la base dieci e un limite piuttosto pesante: **una sola cifra in base 10**.
- Se consideriamo solo i numeri non negativi, possiamo svolgere alcune operazioni senza problemi: 3+5=8. Però diverse combinazioni si comportano in modo strano: 8+3=1! Perché? Semplicemente perché il risultato sarebbe 11, ma il riporto (*carry* in inglese) viene perso, dato che non possiamo utilizzare più di una cifra e teniamo solo quella meno significativa (quella più a destra).
- Nasce il problema dell'*overflow*, ovvero dell'impossibilità di rappresentare il risultato di un'operazione.

Rappresentazione con un numero finito di cifre

- Osserviamo quindi che possiamo rappresentare i numeri in una sorta di sistema circolare, che dopo il 9 torna a zero. Un po'come accade per gli angoli dove 360° è esattamente 0°.
- Le operazioni aritmetiche di somma e sottrazione quindi quando «sforano», ritornano a 0 e continuano, o in senso opposto sotto allo zero tornano a 9.

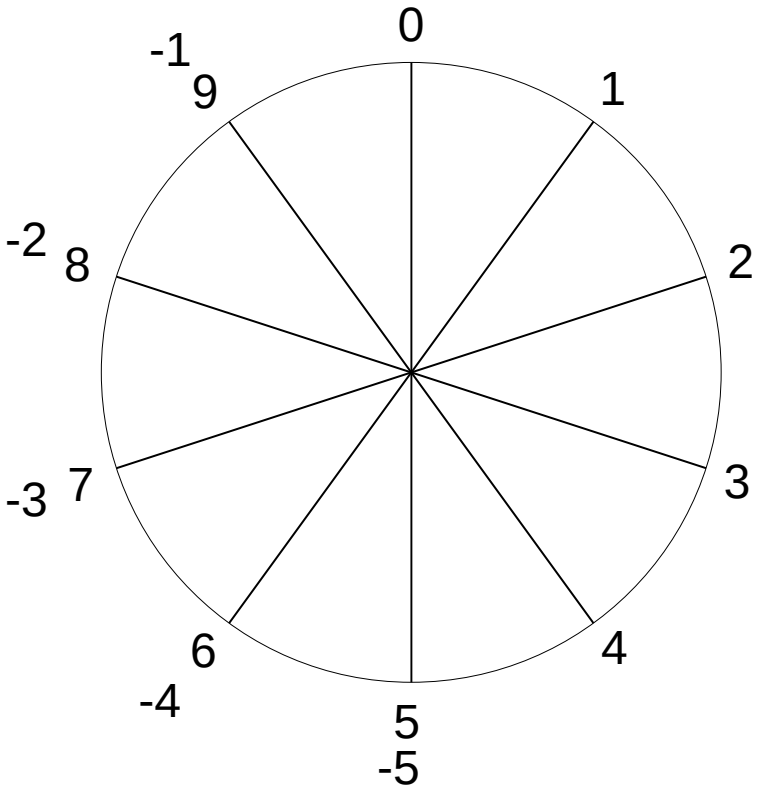


Rappresentazione con un numero finito di cifre

- Quando le cifre sono finite, diventa possibile chiedersi quale sia il più grande numero non negativo rappresentabile e questo ovviamente dipende dalla base e dal numero di cifre.
- Se consideriamo i numeri in base dieci, è facile capire che con una cifra possiamo rappresentare 10 numeri (0-9), con due cifre 100 numeri (00-99), con tre cifre 1000 numeri (000-999) e così via.
- In generale con n cifre in base b possiamo rappresentare b^n diverse combinazioni e se interpretiamo queste come valori interi non negativi, questi sono i numeri da 0 a $b^n - 1$.
- Ad esempio con 4 bit rappresentiamo i numeri da $(0000)_2$ a $(1111)_2$, ovvero da 0 a 15.

Numeri negativi

- Non è però necessario interpretare le combinazioni di n cifre in una certa base come interi non negativi. Possiamo dare una interpretazione logica diversa per i valori visti.
- Volendo introdurre i valori negativi, possiamo immaginare che decrementando di uno lo zero il risultato sia -1. E decrementando ulteriormente si ottenga -2 e così via.
- Facendo questa associazione per metà dei valori (quindi da 5 a 9) e lasciando invece inalterata l'interpretazione per l'altra metà (da 0 a 4) otteniamo uno schema come quello accanto.
- In questo modo l'operazione 8+3=1, che sembrava assurda, viene interpretata come -2+3=1 che è invece molto comune!



Rappresentazione in complemento alla base

- Il ragionamento seguito, ci consente di rappresentare i valori negativi utilizzando solo i simboli della base prescelta (quindi niente simbolo «meno») e di eseguire le operazioni di somma e sottrazione sempre allo stesso modo, ignorando quindi il segno degli operandi.
- Come possiamo estendere questo a più cifre? Il valore utilizzato per i numeri da 5 a 9 è ottenibile sottraendo 10 al valore nominale del simbolo, ovvero effettuando il *complemento alla base*. Se utilizzassimo 2 cifre, il valore 99 dovrebbe corrispondere a -1, quindi dovremmo sottrarre 100.
- Possiamo generalizzare a n cifre e ad una generica base b, con la seguente relazione:

$$v(x) = \begin{cases} x & x < b^n/2 \\ x - b^n & x \geq b^n/2 \end{cases}$$

- dove $v(x)$ è il valore che diamo al numero x. Al contrario

$$x = \begin{cases} v(x) & v(x) \geq 0 \\ v(x) + b^n & v(x) < 0 \end{cases}$$

- ovvero se vogliamo rappresentare un valore positivo utilizziamo la sua rappresentazione, altrimenti sommiamo al numero b^n.

Complemento a 2

- Lavorando con i numeri binari con numero di cifre finito, il complemento a 2 è la rappresentazione più usata.
- Vale sempre la definizione data precedentemente e quindi, facendo un esempio a 4 bit:

codifica a 4 bit	valore come numero non negativo	valore in complemento a 2
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Peculiarità della base 2

- Consideriamo la definizione del valore in complemento alla base:

$$v(x) = \begin{cases} x & x < b^n/2 \\ x - b^n & x \geq b^n/2 \end{cases}$$

- Se $b = 2$

$$v(x) = \begin{cases} x & x < 2^n/2 \\ x - 2^n & x \geq 2^n/2 \end{cases} = \begin{cases} x & x < 2^{n-1} \\ x - 2^n & x \geq 2^{n-1} \end{cases}$$

- Se indichiamo le cifre binarie di x con $\hat{x}_i$, è chiaro che

$$x < 2^{n-1} \Leftrightarrow \hat{x}_{n-1} = 0$$

- Quindi sostituendo nella definizione precedente

$$v(x) = \begin{cases} x & \hat{x}_{n-1} = 0 \\ x - 2^n & \hat{x}_{n-1} = 1 \end{cases}$$

- Utilizziamo ora la rappresentazione di x in base 2 con n cifre:

$$x = \sum_{i=0}^{n-1} \hat{x}_i 2^i = \hat{x}_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} \hat{x}_i 2^i$$

- Sostituendo nella definizione del valore in complemento a due si ottiene (segue)

Peculiarità della base 2

$$v(x) = \begin{cases} 0 \cdot 2^{n-1} + \sum_{i=0}^{n-2} \hat{x}_i 2^i & \hat{x}_{n-1} = 0 \\ 1 \cdot 2^{n-1} + \left(\sum_{i=0}^{n-2} \hat{x}_i 2^i \right) - 2^n & \hat{x}_{n-1} = 1 \end{cases}$$

- Nel secondo caso si ha che:
$$1 \cdot 2^{n-1} - 2^n = 1 \cdot 2^{n-1} - 2 \cdot 2^{n-1} = -1 \cdot 2^{n-1} = 1 \cdot (-2^{n-1})$$
- Ovviamente nel primo caso possiamo sostituire:
$$0 \cdot 2^{n-1} = 0 \cdot (-2^{n-1})$$
- Quindi
$$v(x) = -\hat{x}_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} \hat{x}_i 2^i$$
- Nel caso della base due, quindi, la rappresentazione in complemento a 2 con n cifre consiste nel far valere la cifra binaria più significativa -2^{n-1}, senza modificare l'interpretazione delle altre cifre binarie.

Conversioni

- Consideriamo il numero binario a 8 bit 1111.1111, che sappiamo essere rappresentato in complemento a 2:

$$(1111.1111)_2 = -128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = -1$$

- come ovvio, dato che è il massimo valore rappresentabile con 8 bit, ovvero quello che se incrementato di 1 torna a 0 (richiederebbe 9 bit per effettuare l'incremento).

- Come possiamo ottenere il valore assoluto di un numero rappresentato in complemento a 2? Esistono due algoritmi equivalenti (non dimostriamo):

1. Si invertono tutti i bit del numero (si esegue il *complemento a 1*) e poi si somma 1.
2. Si copiano tutti i bit partendo dal meno significativo fino al primo bit uguale a 1 incluso. Tutti gli altri vanno invertiti. (detto diversamente: si invertono tutti i bit a sinistra dell'1 meno significativo)

Esempi

- Come si scrive in binario a 8 bit in complemento a 2 il numero -59? Partiamo rappresentando il suo valore assoluto:

59	1
29	1
14	0
7	1
3	1
1	1
0	

Quindi $59 = (0011.1011)_2$.

- Applichiamo il primo algoritmo indicando con la linea sopra al numero il complemento a 1: $-59 = \overline{(0011.1011)_2} + 1 = (1100.0100)_2 + 1 = (1100.0101)_2$. È immediato verificare che abbiamo invertito tutti i bit a sinistra dell'1 meno significativo.
- Quanto vale $(1100.0101)_2$? $-128 + 64 + 4 + 1 = -59$, come richiesto.

Numeri in virgola fissa

- Che cosa succede ai numeri con la virgola quando limitiamo il numero di cifre disponibili?
- Fondamentalmente nulla, a parte il fatto che dobbiamo decidere quali bit dedichiamo alla parte intera e quali a quella frazionaria.
- Supponiamo ad esempio di utilizzare 8 bit per numero e di rappresentare in complemento a due i valori negativi, riservando 4 bit per la parte intera e 4 per quella frazionaria. Il nostro numero è suddiviso così:

	Parte intera				Parte frazionaria			
Potenza del 2	3	2	1	0	-1	-2	-3	-4
Valore	-8	4	2	1	0,5	0,25	0,125	0,0625
Cifre binarie	0	0	1	0	0	1	1	0
Valore x cifra	0	0	2	0	0	0,25	0,125	0

Valore decimale	2,375

- Dunque $(0010{,}0110)_2 = 2{,}375$.

Numeri in virgola fissa

- Quali sono i limiti della nostra rappresentazione? Il numero più grande è $(0111{,}1111)_2 = 7{,}9375$, mentre il numero positivo più piccolo è $(0000{,}0001)_2 = 0{,}0625$. Il valore più negativo sarà $(1000{,}0000)_2 = -8{,}0$.
- Non è un intervallo di valori sorprendente. E se utilizzassimo 32 bit? Dobbiamo comunque decidere quanti bit dedicare alla parte intera e quanti a quella frazionaria.
- Se facciamo metà e metà, ovvero 16 bit per parte, il numero più grande diventa $32767{,}999984741210937 5$, il numero positivo più piccolo è $0{,}000015258789062 5$. Questo è anche il minimo incremento possibile tra un numero e il successivo.
- L'intervallo non è enorme, infatti il numero più grande non consente di memorizzare neppure il valore 100000 e per una macchina che deve fare calcoli non sembra un gran ché.
- Anche la precisione non è poi fenomenale: meno che 5 cifre decimali.
- Il problema è che utilizziamo la quinta cifra decimale anche quando parliamo di valori intorno al 10000!
- Serve una rappresentazione più flessibile.

Numeri in virgola mobile

- La rappresentazione dei numeri in virgola mobile è in relazione con la notazione scientifica (es. $1.2 \times 10^2 = 120$)
- Il concetto di base è che in questo modo utilizziamo le cifre significative dove ci servono effettivamente.
- Se consideriamo una notazione decimale in cui il numero di cifre sia 1 prima del punto decimale, 3 dopo il punto decimale e 2 cifre all'esponente, possiamo rappresentare valori in questo range:

  -9.999×10^{99} ... -0.001×10^{-99} ,0, 0.001×10^{-99} ... 9.999×10^{99}
- Complessivamente quindi dobbiamo memorizzare il segno del numero, la parte intera (1 cifra), la parte decimale (3 cifre), il segno dell'esponente, il modulo dell'esponente (2 cifre).
- È chiaro che l'operazione 10000 + 1, con queste limitazioni, non è eseguibile correttamente: $10000 = 1.000 \times 10^{04}$, $1 = 1.000 \times 10^{00}$,
- Per eseguire la somma è necessario portare 1 alla potenza 10^4 e quindi diventerebbe: $1 = 0.0001 \times 10^{04}$, ma con i nostri limiti → 0.000×10^{04} e dunque 10000+1=10000!

Lo standard IEEE 754

- Lo **standard IEEE per il calcolo in virgola mobile** (**IEEE 754**) (ufficialmente: **IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985)** è lo standard più diffuso nel campo del calcolo automatico. Questo standard definisce il formato per la rappresentazione dei numeri in virgola mobile (compreso ±0 e i numeri denormalizzati; gli infiniti e i NaN, "*not a number*"), ed un set di operazioni effettuabili su questi. Specifica inoltre quattro metodi di arrotondamento e ne descrive cinque eccezioni.
- Esistono in questo standard quattro formati per i numeri in virgola mobile: a *precisione singola* (32 bit), *precisione doppia* (64 bit), *precisione singola estesa* (≥ 43 bit), raramente usato, e *precisione doppia estesa* (≥ 79 bit), supportata solitamente con 80 bit. La precisione singola è il minimo richiesto dallo standard, gli altri sono opzionali.

Rappresentazione

- Un numero in virgola mobile, secondo lo standard IEEE è rappresentato su parole di 32, 64 o 128 bit divisi in tre parti:
  - un bit di **segno** *s*;
  - un campo di **esponente** *e*;
  - un campo di **mantissa** *m*

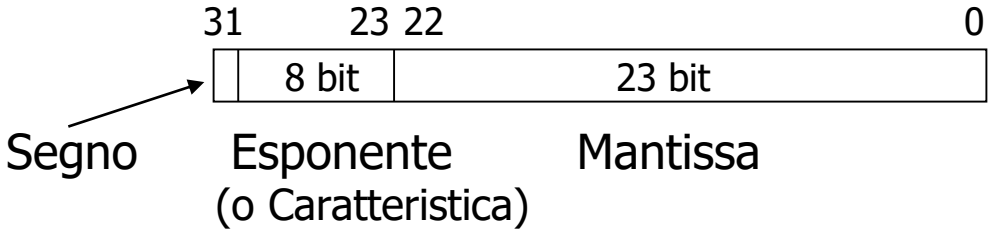
  in questo ordine. Gli *n* bit di una stringa sono indicizzati in modo decrescente con numeri interi da 0 a *n*-1. In un numero in questo standard, l'importanza del bit decresce col suo indice.

Name	Common name	Base	Digits	E min	E max	Decimal digits	Decimal E max
binary32	Single precision	2	23+1	−126	+127	7.22	38.23
binary64	Double precision	2	52+1	−1022	+1023	15.95	307.95
binary128	Quadruple precision	2	112+1	−16382	+16383	34.02	4931.77

Decimal digits è *digits* × $\log_{10}$ *base*, circa la precisione in cifre decimali.
Decimal E max è *Emax* × $\log_{10}$ *base*, il massimo esponente in decimale.

Numeri in virgola mobile

- I numeri in singola precisione riservano 1 bit per il segno (come tutti gli altri), 8 bit per l'esponente e 23 bit per la mantissa.
- Il campo esponente E è quindi un valore da 0 a 255, ma gli estremi vanno trattati come «casi particolari»
- Nel caso standard, $1 \leq E \leq 254$, si parla di *numeri normalizzati*.



- Il valore dei numeri normalizzati si può calcolare come
$$(-1)^s \times 1.m \times 2^{E-127}$$
- In sostanza, il valore dell'esponente è memorizzato in *eccesso 127*. Si dice anche che al campo esponente E viene applicato un *bias* di 127.
- Il valore dell'esponente (che indicheremo con la lettera minuscola «*e*») è quindi limitato nel range $-126 \leq e \leq 127$.

Numeri denormalizzati e altri casi

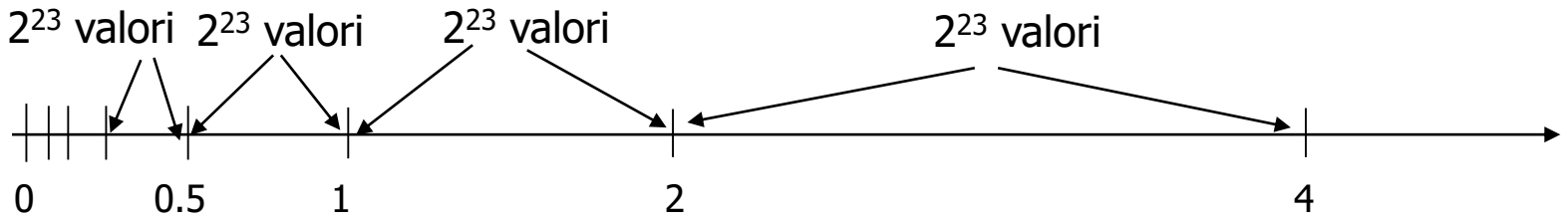
- Quando $E = 0$ si hanno i *numeri denormalizzati*, per i quali il valore si può calcolare come
$$(-1)^s \times 0.m \times 2^{-126}$$
- In questo modo è possibile rappresentare lo zero (tutti i bit a 0) e altri valori nel suo intorno con precisione di $2^{-149} \cong 1{,}401 \times 10^{-45}$.
- Quando invece $E = 255$ si hanno i valori speciali, ovvero gli infiniti e i NaN (Not a Number).
- Il valore di m distingue tra i due casi: se $m = 0$ il numero è un infinito, ovvero i due valori possibili sono (in esadecimale) 7f800000 ($+\infty$) e ff800000 ($-\infty$).
- Se invece $m \neq 0$ il numero è un NaN, quindi potrebbe essere trattato diversamente.
- Ad esempio 1.0/0.0→$+\infty$, mentre 0.0/0.0→NaN, come anche $\sqrt{-1}$.
- Il compilatore e le librerie Microsoft indicano il NaN con -1.#IND0000 e usano il valore ffc00000.

Numeri in singola precisione

- Il numero più grande rappresentabile in singola precisione è dato da

$$2^{127} + \sum_{i=-1}^{-23} 2^{127+i} = \sum_{i=104}^{127} 2^i = 2^{128} - 2^{104}$$

  che fa 340282346638529000000000000000000000, o in modo più compatto $3.40282347 \times 10^{38}$. In esadecimale è 7f7fffff.
- Il più piccolo numero positivo, come detto è invece $2^{-149} \cong 1{,}401 \times 10^{-45}$
- In totale si rappresentano 2^{32} numeri distinti, metà positivi, metà negativi.
- Circa metà dei numeri sono compresi fra -1 e 1 ($e < 0$, ovvero $E < 127$).
- I numeri non sono distribuiti uniformemente sulla retta dei numeri reali, ma la precisione si dimezza ad ogni intervallo di potenze del due:



Codifica dei caratteri alfabetici

- I numeri sono certamente importanti, ma i calcolatori devono anche comunicare con gli esseri umani, consentire loro di comunicare e rappresentare le informazioni scritte diventa quindi fondamentale.
- Inoltre è necessario disporre di un modo per rappresentare i risultati di quanto elaborato, mostrandolo in modo comprensibile ad un utente.
- Pertanto uno dei tipi di informazione che deve essere possibile codificare è quella testuale. Si potrebbe procedere in molti modi con questo scopo, ma il primo standard internazionale è statunitense e rappresenta il testo come sequenza di simboli letterali, fondamentalmente le nostre «lettere».
- Questo non è scontato, in giapponese ad esempio nel sistema di scrittura hiragana, ogni carattere corrisponde ad un'intera sillaba e quindi si potrebbe codificare la parola Tsukuba (つくば), un polo tecnologico vicino Tokyo, con tre codici, uno per つ (tsu), uno per く (ku) e uno per ば (ba).

Il codice ASCII

- Il primo standard internazionale, ancora oggi supportato da tutti i sistemi di calcolo, è l'*American Standard Code for Information Interchange* (ASCII).
- È uno schema per la codifica dei caratteri basato originariamente sull'alfabeto inglese.
- Questo utilizza interi non negativi in base due con 7 bit per codificare 128 diversi «caratteri»:
  - i numeri da «0» a «9»
  - le lettere da «a» a «z» e da «A» a «Z»
  - alcuni simboli di punteggiatura
  - alcuni codici di controllo utilizzati nelle telescriventi
  - un carattere «spazio»
- La pronuncia corretta assomiglia a come in Italia pronunciamo la razza di cani Husky, anche se in Italia la coppia «sc» viene pronunciata come in «sciare», come al solito leggendolo come è scritto.

La tabella dei caratteri ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

I codici

- Le origini delle scelte fatte per questo standard sono un compromesso tra diversi produttori di sistemi per la comunicazione degli anni ’60/’70 pertanto molti dei codici di controllo non hanno più alcuna utilità.
- Certamente è possibile notare che i codici sono raggruppati utilizzando i tre bit più significativi (una h indica che il numero è esadecimale nel seguito):
  - I codici di controllo vanno da 00h a 1Fh
  - Lo spazio e gran parte della punteggiatura vanno da 20h a 2Fh
  - I numeri vanno da 30h a 3Fh
  - Le maiuscole da 40h a 5Fh
  - Le minuscole da 60h a 7Fh
- L’elenco appena fatto è solo approssimativo e per ragioni storiche, ad esempio le lettere non partono da 40h e 60h, ma da 41h e 61h.
- Inoltre i gruppi dei numeri e delle lettere non sono «pieni» e quindi sono stati riempiti con ulteriori segni di punteggiatura.
- Anche il codice 7Fh (DEL) è un vecchio codice di controllo. Questo e gli altri codici di controllo non erano pensati per essere visualizzati.

Codici di controllo

- I 33 codici di controllo, sono descritti nel dettaglio nella pagina http://en.wikipedia.org/wiki/ASCII e hanno principalmente un interesse storico.
- Rimangono importanti i caratteri:
  - 00h (NUL - Null character), utilizzato per indicare la fine di sequenze di caratteri.
  - 09h (HT - Horizontal Tab), utilizzato come separatore di dati
  - 0Ah (LF - Line feed), usato per indicare l'«a capo»
  - 0Dh (CR - Carriage return), usato per indicare l'«a capo»
- In particolare sotto Unix e i suoi derivati l'«a capo» è indicato da LF, sotto DOS/Windows dalla coppia CR LF, anticamente sotto i sistemi Apple dal solo CR. Attualmente anche i sistemi Apple utilizzano solo LF e «Blocco Note» è l'unico programma (al mondo?) che richiede assolutamente la coppia CR LF per andare a capo.

Tabella ASCII estesa

- Poco dopo la sua introduzione, la prassi volle che l'unità fondamentale per la gestione di dati binari fosse a 8 bit. Ogni produttore quindi estese lo standard con altri 128 codici, sfruttandoli per fornire caratteri di altre lingue. Qui di seguito è visualizzata una rappresentazione dei codici estesi utilizzati nella codepage 437, quella utilizzata dall'MS-DOS originale in inglese.

128	Ç	144	É	160	á	176	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	161	í	177	▒	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	162	ó	178	▓	195	├	211	╙	227	π	243	≤
131	â	147	ô	163	ú	179	│	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	164	ñ	180	┤	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	165	Ñ	181	╡	198	╞	214	╓	230	µ	246	÷
134	å	150	û	166	ª	182	╢	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	167	º	183	╖	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	168	¿	184	╕	201	╔	217	┘	233	⊛	249	·
137	ë	153	Ö	169	_	185	╣	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	170	¬	186	║	203	╦	219	▉	235	δ	251	√
139	ï	156	£	171	½	187	╗	204	╠	220	▄	236	∞	252	_
140	î	157	¥	172	¼	188	╝	205	═	221	▌	237	ϕ	253	²
141	ì	158	_	173	¡	189	╜	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	174	«	190	╛	207	╧	223	▀	239	∩	255	
143	Å	192	└	175	»	191	┐	208	╨	224	α	240	≡		

Unicode

- Al mondo non esistono solo gli americani!
- Appena la diffusione dei calcolatori raggiunse gli stati al di fuori degli USA, divenne indispensabile gestire i caratteri con accenti e tutti i sistemi linguistici come cinese, giapponese o una delle tante lingue dell'India.
- In India sono state riconosciute 1652 lingue! Di queste però «solo» 30 sono parlate da più di un milione di persone. Ci sono 9 diversi insiemi di caratteri: Bengali, Devanagari, Gujarati, Gurmukhi, Kannada, Malayalam, Oriya, Tamil, and Telugu.
- A partire dal 1987 apparve evidente che tutte le soluzioni basate su set di caratteri personalizzati per regione fossero inutilizzabili in pratica. Alla Xerox, proposero quindi un sistema che utilizzava 16 bit per carattere.
- Questo venne standardizzato nel 1991 con la pubblicazione dello standard Unicode. Consentiva di specificare quasi 65536 diversi caratteri.
- Già nel 1996 però venne introdotto l'Unicode 2.0 che consentiva di superare questo limite. Al momento (Settembre 2019) siamo all'Unicode 12.1 che contiene 137994 caratteri da 150 grafie diverse.

Codifica dei caratteri Unicode

- Il sistema Unicode è molto complesso e solo in rarissimi casi ci si addentra al suo interno, dato che esistono caratteri indicati con un solo codice, caratteri composti (caratteri base e modificatori), legature e altri accrocchi difficilmente comprensibili.
- È però importante sapere che in Unicode sono presenti 1.114.112 *code points*, ovvero **possibili** codici che rappresentano caratteri o variazioni su caratteri.
- Esistono diverse codifiche e la prima che fu proposta è ora nota come UCS-2 da Universal Character Set e 2 byte per simbolo.
- È molto semplice, facilmente comprensibile e, purtroppo, inadatta ad applicazioni moderne, dato che non è in grado di rappresentare tutti i caratteri già presenti in Unicode. Funziona solo per il *Basic Multilingual Plane*, ovvero il primo set di caratteri previsto da Unicode.
- Appena ci si accorse che 65.536 caratteri (code points) non erano sufficienti si cercarono soluzioni alternative.
- Attualmente il riferimento sono gli *Unicode Transformation Format* (UTF).

Unicode Transformation Format (UTF)

- Per sopperire alle limitazioni delle codifiche a lunghezza fissa precedenti furono introdotti 3 formati di codifica noti come
  - UTF-8
  - UTF-16
  - UTF-32
- Il più comune e da alcuni considerato l'unico da utilizzare è l'UTF-8.
- Ufficializzato nel 1993, le idee alla base dell'UTF-8 sono
  - compatibilità con l'ASCII (da 0 a 127)
  - self-synchronization o auto-sincronizzazione: se si parte a metà di un codice, non è possibile interpretarlo in modo errato

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

UTF-16 e UTF-32

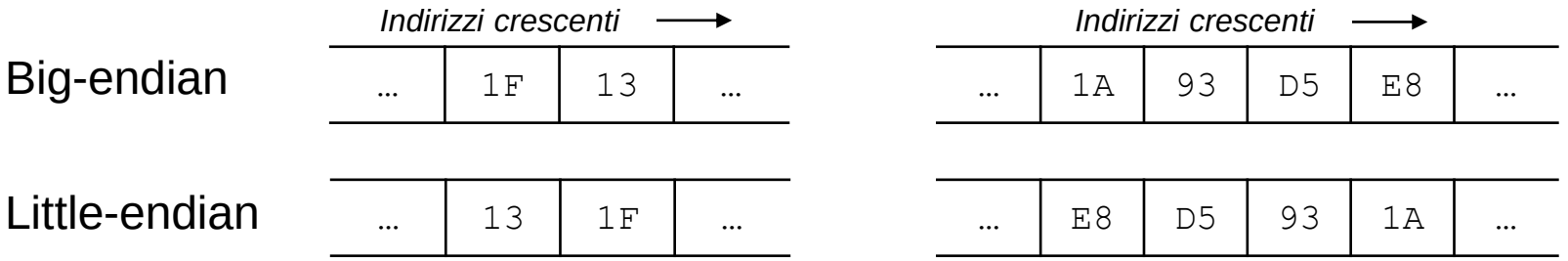
- Le altre due versioni di UTF sono meno comuni e non li vedremo in dettaglio.
- È sufficiente sapere che il primo utilizza 16 bit per rappresentare i code point Unicode da U+0000 a U+D7FF e da U+E000 a U+FFFF
- Quelli da U+D800 a U+DFFF non sono valori validi per Unicode e quindi vengono utilizzati per indicare che il codice successivo è la continuazione di questo.
- UTF-32 utilizza 32 bit per carattere ed è utilizzato sostanzialmente solo internamente da alcuni sistemi per semplificare la gestione della rappresentazione grafica. Nello standard HTML5 si dice esplicitamente che non deve essere utilizzato.
- Rispetto a UTF-8 hanno il problema della *endianness*, ovvero dell'ordine in cui i valori composti da più byte sono rappresentati e memorizzati.

Endianness (1)

- Con il termine *endianness* si intende l'ordine in cui vengono disposti **i byte** di un valore costituito da più byte su un dispositivo di memorizzazione o trasmissione (memoria RAM, file su disco, rete).
- Esistono due tipi di endianness:
  - **big-endian**: il byte meno significativo è all'indirizzo più grande
  - **little-endian**: il byte meno significativo è all'indirizzo più piccolo

  Esempio:

  Consideriamo i due numeri 7955 e 445896168, il primo rappresentato in memoria con 2 byte mentre il secondo con 4 byte e in esadecimale valgono rispettivamente 1F13h e 1A93D5E8h.

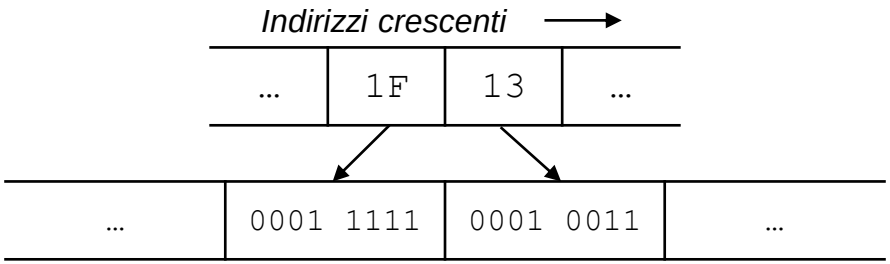


- I processori Intel lavorano in little-endian mentre altre architetture (meno diffuse al giorno d'oggi) lavorano in big-endian.

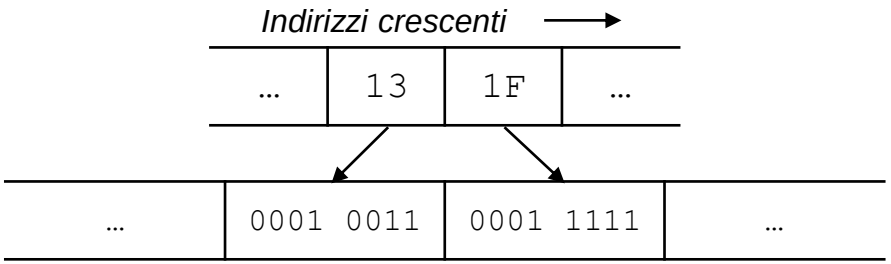
Endianness (2)

- ATTENZIONE: quando si parla di little-endian e big-endian si parla solamente dell'ordine dei byte!
- Consideriamo nuovamente il numero 7955 = 1F13h e le sue rappresentazioni in little-endian e big-endian:

Big-endian



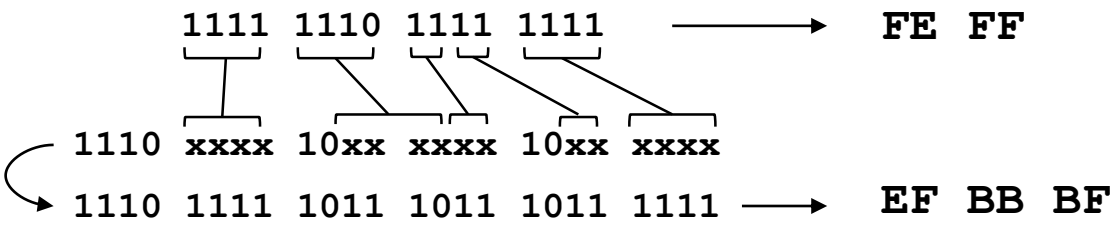
Little-endian



- L'ordine dei bit all'interno dei byte resta invariato!
- Convertire un valore tra little-endian e big-endian significa invertire l'ordine dei byte, non delle cifre (esadecimali o binarie)!

Byte Order Mark (BOM)

- La differenza tra *little-endian* e *big-endian* si può trovare proprio nello standard Unicode. Nella codifica UTF-16, dove ogni simbolo è rappresentato con 2 byte, il cosiddetto «Byte Order Mark» viene posto all'inizio di uno stream o di un file di testo ed è utilizzato per stabilirne la endianness.
- Il BOM è il valore FEFFh, l'ordine di questi due byte stabilisce la endianness:
  - Se viene letto FEFFh i simboli successivi sono rappresentati in big-endian.
  - Se viene letto FFFEh i simboli successivi sono rappresentati in little-endian.
- Nella codifica UTF-8 il BOM viene codificato con 3 byte:



- Non avendo senso parlare di endianness nell'UTF-8, il BOM è opzionale ed è utilizzato solamente per indicare che i simboli successivi sono rappresentati tramite la codifica UTF-8.

Un carattere famoso: క్ష

- Nel febbraio 2018 il «carattere» Telugu rappresentato qui sopra ha mandato in tilt numerosi iPhone, quando visualizzato nell'area delle notifiche.
- Ma che caratteristiche ha questo «carattere»?

Browser	Character	Name	# Fonts
జ	U+0C1C	TELUGU LETTER JA	8
◌్	U+0C4D	TELUGU SIGN VIRAMA halant (the preferred name)	8
ష	U+0C1E	TELUGU LETTER NYA	8
	U+200C	ZERO WIDTH NON-JOINER	286
◌ా	U+0C3E	TELUGU VOWEL SIGN AA	8