Rappresentazione dell’informazione

Architettura degli Elaboratori

AA 2019-20

Rappresentazione dei numeri naturali

- Possibili codifiche dei numeri naturali 0,1,2,…
- Un codice ovvio è dato dal sistema di numerazione decimale basato su A={‘0’,…,’9’}.
- Tale sistema è un sistema posizionale ovvero ad ogni cifra di una parola è assegnato un peso differente a seconda della posizione nella sequenza. Ad esempio dato “4456”
  - ‘6’ rappresenta sei unità (cifra meno significativa)
  - ‘5’ rappresenta cinque decine
  - ‘4’ rappresenta quattro centinaia
  - ‘4’ rappresenta quattro migliaia (cifra più significativa)

Rappresentazione in base 10

- Decomponendo in potenze di 10, il numerale 1024 rappresenta il numero
$$1 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0$$
- Generalizzando, un numerale $c_{m-1}c_{m-2}\cdots c_0$ rappresenta
$$\sum_{i=0}^{m-1} c_i \cdot 10^i$$
- Il sistema decimale è quindi una codifica posizionale su base 10. Tuttavia non è l’unica, ad esempio i babilonesi utilizzavano un sistema di numerazione su base 60 (sessagesimale)
- I dispositivi digitali per elaborare e memorizzare l’informazione possono trovarsi in due possibili stati che rappresentano la cifra ‘0’ e la cifra ‘1’. Il sistema corrispondente è quello in base 2 (binario)

Rappresentazione in una base generica

- Ad ogni naturale b>1 corrisponde una codifica in base b.
- L'alfabeto A_b consiste in b simboli distinti che corrispondono ai numeri 0,1,…, b-1.
- Analogamente al sistema decimale, un numerale di cifre di A_b rappresenta il numero $s_{m-1} \dots s_0$

$$\sum_{i=0}^{m-1} s_i \cdot b^i$$

- Consideriamo ad esempio il sistema ottale (base 8).

  A_8 consiste nelle cifre '0','1',…,'7'

$$13 \longrightarrow 1 \cdot 8^1 + 3 \cdot 8^0 = 8 + 3 = 11$$

$$5201 \longrightarrow 5 \cdot 8^3 + 2 \cdot 8^2 + 0 \cdot 8^1 + 1 \cdot 8^0 = 5 \cdot 8^3 + 2 \cdot 8^2 + 1 \cdot 8^0 = 2560 + 128 + 1 = 2689$$

Rappresentazione in una base generica

A questo punto potrebbero sorgere delle ambiguità. Se qualcuno vi dicesse: 11 è un numero pari. Pensereste che sia impazzito. Lui potrebbe ribattere: certo! infatti in base 5

$$11 \longrightarrow 1 \cdot 5^1 + 1 \cdot 5^0 = 5 + 1 = 6$$

E' chiaro che bisogna mettersi d'accordo su quale sistema di numerazione si adotta di volta in volta. Per questo useremo delle notazioni standard

- n denota un (generico) numero naturale, a prescindere dalla sua rappresentazione
- n_b denota un naturale rappresentato in base b
- Una sequenza di cifre decimale rappresenta un particolare naturale espresso in base dieci. Ad esempio 236 denota il numero duecentotrentasei
- Una sequenza di cifre seguite da un pedice b rappresenta un numero espresso in base b.
- esempio:

$$1001_2 = 2^3 + 1 = 9$$

Basi maggiori di 10

- Per basi b < 10 possiamo chiaramente ri-usare le usuali cifre ‘0’,…,’9’. Ad esempio, la codifica in base 6 utilizza le cifre ‘0’,…,’5’ mentre quella in base 3 le cifre ‘0’, ‘1’ e ‘2’ e così via.
- E per basi b>10?
  - Si prendono in prestito le lettere dell’alfabeto
  - Per b=16 le cifre adottate sono ‘0’,…,’9’,’a’,…,’f’, dove:

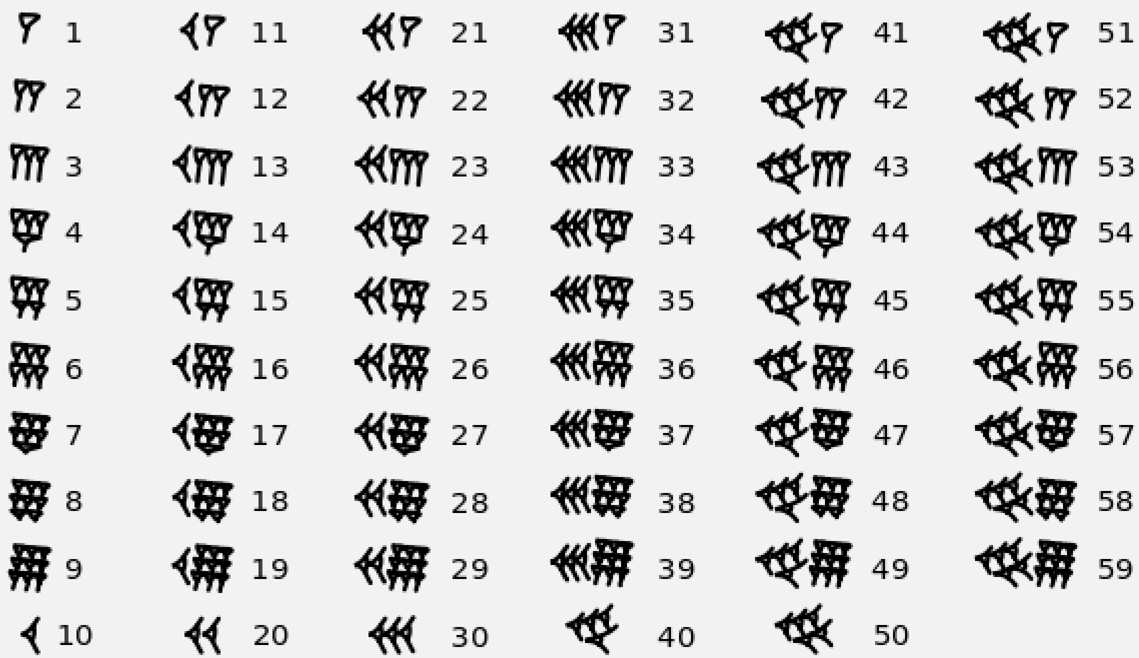
$$\begin{array}{ll} a_{16} = 10 & b_{16} = 11 \\ c_{16} = 12 & d_{16} = 13 \\ e_{16} = 14 & f_{16} = 15 \end{array}$$

Esempio:

$$b3c_{16} = 11 \cdot 16^2 + 3 \cdot 16^1 + 12 \cdot 16^0 = 2816 + 48 + 12 = 2876$$

Basi maggiori di 10

- E se le lettere dell'alfabeto non dovessero bastare?



Lunghezza di n rispetto ad una base

- Chiamiamo lunghezza di n rispetto a b il numero di cifre che occorrono per rappresentare n in base b

  - la lunghezza di 101 rispetto a 2 è 7: $1100101_2=101$
  - la lunghezza di 101 rispetto a 10 è 3: $101_{10} = 101$
  - la lunghezza di 101 rispetto a 16 è 2: $65_{16}=101$

- La lunghezza di un numerale decresce al crescere della base di codifica

Valore massimo rappresentabile

Riferendoci alla base 10

- Con 1 cifra rappresentiamo i numeri da 0 a 9
- Con 2 cifre i numeri da 0 a 99
- Con 3 cifre i numeri da 0 a 999
- Con m cifre i numero da 0 a 10^m-1

Quindi se indichiamo con v_{max} il maggior numero rappresentabile con m cifre in base 10 abbiamo

$$v_{max} = 10^m - 1$$

Valore massimo rappresentabile

- Per una base diversa da 10, quale è il massimo valore rappresentabile con m cifre?
- Consideriamo, ad esempio, il caso b=2 e m=4, il massimo valore rappresentabile corrisponde al numerale 1111

$$1111_2 = 2^3 + 2^2 + 2^1 + 2^0 = 15 = 2^4 - 1$$

- Per una generico b e m il maggior numero rappresentabile si ottiene concatenando m volte la cifra “più alta” (b-1). Quindi:

$$v_{max} = (b-1)b^{m-1} + (b-1)b^{m-2} + \ldots + (b-1)b^1 + (b-1)b^0 =$$

$$= (b \cdot b^{m-1} - b^{m-1}) + (b \cdot b^{m-2} - b^{m-2}) + \ldots + (b \cdot b^1 - b^1) + (b \cdot b^0 - b^0) =$$

$$= b^m - b^{m-1} + b^{m-1} - b^{m-2} + \ldots + b^2 - b + b - 1 = b^m - 1$$

Cifre necessarie per rappresentare n

- Nella slide precedente avevamo il numero di cifre m e volevamo sapere quale è il massimo numero rappresentabile in una certa base b.
- Consideriamo il problema inverso: abbiamo un valore n e ci chiediamo quante cifre m occorrono per rappresentarlo.
- Chiaramente il massimo valore rappresentabile con m cifre dovrà essere maggiore o uguale a n

$$b^m - 1 \geq n$$
$$b^m \geq n + 1$$
$$m \geq \log_b(n+1)$$

- In particolare cerchiamo il più piccolo m tale che $m \geq \log_b(n+1)$

$$m = \lceil \log_b(n+1) \rceil$$

Cifre necessarie per rappresentare n

- Notazione: dato un reale x
  - $\lceil x \rceil$ denota l'approssimazione intera per eccesso
  - $\lfloor x \rfloor$ denota l'approssimazione intera per difetto
- Esempi:
  - $\lceil 14.78 \rceil = 15$
  - $\lfloor 9.88 \rfloor = 9$
- In qualche testo potrete trovare $m = \lfloor \log_b n \rfloor + 1$
  - C'è un errore ?

→ per ogni n e b:

$$\lceil \log_b(n+1) \rceil = \lfloor \log_b n \rfloor + 1$$

Relazione fra due basi diverse

- Considieriamo due sistemi di numerazione nelle basi a e b. Dato un naturale n, quale relazione intercorre fra m_a e m_b?
- Per ogni base c $\lfloor \log_c n \rfloor + 1 - \log_c n \leq 1$
- Quindi considereremo le approssimazioni

$$m_a \simeq \log_a n$$
$$m_b \simeq \log_b n$$

- Ricordando che: $\log_b n = \log_a n * \log_b a$
- Il rapporto fra le lunghezze m_b e m_a è costante (indipendente da n) e pari a $\log_b a$:

$$m_b = \log_b n = \log_a n * \log_b a = m_a * \log_b a$$

$$m_b / m_a = \log_b a$$

Digressione: base unaria

- Ricordate che quando abbiamo introdotto i sistemi di numerazione abbiamo assunto la base $b \geq 2$.
- Perché non è possibile considerare una numerazione unaria? Certo! Ma ci sono delle controindicazioni…
    - La base unaria consta di una solo cifra I
    - Una I rappresenta 0, due II 1, …, n+1 I rappresentano n
    - $IIIIII_1 = 5$
    - Notate che a prescindere dalla posizione ogni I vale 1, quindi questo sistema non è posizionale
    - Non potrebbe essere altrimenti visto che 1 elevato ad una qualsiasi potenza è sempre uguale a 1!

Digressione: base unaria

Andiamo a confrontare le lunghezze dei numerali con una base b≥2

$$m_1 = n + 1 = b^{\log_b n} + 1 \simeq b^{m_b} + 1 \simeq b^{m_b}$$

Questo vuol dire che la codifica di n in base unaria cresce esponenzialmente rispetto alla codifica in base b!

La base unaria non è un buon sistema di rappresentazione

Codifica ottimale

- Ritorniamo alle basi ammissibili ($b \geq 2$)
- Abbiamo visto che più grande è la base b, minore è il numero di cifre che occorrono per rappresentare un numero
- Quindi il codice binario è il sistema di codifica meno economico fra quelli ammissibili
  - Per esempio $\log_2 10 \cong 3.32$ questo vuol dire che per rappresentare un numero n in binario occorrono circa il triplo delle cifre che occorrono per rappresentare lo stesso n nel sistema decimale
  - Perché i computer adottano la codifica binaria?

Codifica ottimale

- Non bisogna tener presente solo la lunghezza di codifica ma anche il fatto che un qualsiasi dispositivo che voglia operare in una certa base deve poter rappresentare tutte le cifre di quella base, quindi gli servono b differenti stati.
- Una nozione di costo di una codifica che tiene conto anche di questo fattore è data dal prodotto

$$b \cdot m_b \simeq b \cdot \log_b n$$

- Si può dimostrare che il valore di b che minimizza tale costo è il numero di Nepero e $\cong$ 2.7, quindi le codifiche che si avvicinano di più a tale valore ottimale sono quelle in base 2 e 3

Cambiamento di base

- Problema: dato n rappresentato in base a, quale è la sua rappresentazione in base b?
- Supponiamo di saper fare le quattro operazioni in base a
- L'algoritmo di cambiamento di base consiste nel dividere ripetutamente n (espresso in base a) per b finché il quoziente non risulti uguale a zero. La sequenza di resti ottenuti (compresi tra 0 e b-1) è la codifica dalla cifra meno significativa a quella più significativa di n in base b
- Esempio: codificare 251_{10} in base 3

$$\begin{aligned}
251/3 &= 83 \quad \text{resto: } 2\\
83/3 &= 27 \quad \text{resto: } 2\\
27/3 &= 9 \quad \text{resto: } 0\\
9/3 &= 3 \quad \text{resto: } 0\\
3/3 &= 1 \quad \text{resto: } 0\\
1/3 &= 0 \quad \text{resto: } 1
\end{aligned}$$

$$251_{10} = 100022_3$$

Cambiamento di base

- Esempio: codificare 333_7 in base 9
- Problema: non siamo molto allenati con la divisione in base 7. Meglio affrontare il problema in due passi:
  - Codifico 333_7 in base 10

$$333_7 = 3 \cdot 7^2 + 3 \cdot 7^1 + 3 \cdot 7^0 = 171$$

  - Codifico 171_{10} in base 9

$$\begin{aligned} 171/9 &= 19 \quad \text{resto: } 0 \\ 19/9 &= 2 \quad \text{resto: } 1 \\ 2/9 &= 0 \quad \text{resto: } 2 \end{aligned}$$

$$333_7 = 210_9$$

Codifica binaria e esadecimale

- Abbiamo detto che i componenti digitali operano in codice binario.
- Tuttavia la rappresentazione in binario non è molto human-friendly perché genera codici piuttosto lunghi
  - $599 = 1001010111_2$
- Si potrebbe pensare di usare la nostra base naturale (10). Questo comporta usare il precedente algoritmo per codifica/decodifica
  - Non proprio agevole
  - Non proprio velocissimo
  - Il problema è che 10 non è una potenza di 2

Codifica binaria e esadecimale

- La codifica/decodifica in una base m potenza di 2 permette invece di usare qualche trucchetto che migliora i tempi di codifica e decodifica.
- le cifre utilizzate nel sistema esadecimale sono ‘0’,…,’9’,’a’,…,’f’.
- Inoltre, poiché $16=2^4$ è possibile codificare ogni cifra del sistema esadecimale mediante 4 bit
- Viceversa 4 bit del sistema binario corrispondono ad una cifra esadecimale

Codifica binaria e esadecimale

- Codifica delle cifre esadecimali in binario

$$0 \longrightarrow 0000 \qquad 8 \longrightarrow 1000$$
$$1 \longrightarrow 0001 \qquad 9 \longrightarrow 1001$$
$$2 \longrightarrow 0010 \qquad a \longrightarrow 1010$$
$$3 \longrightarrow 0011 \qquad b \longrightarrow 1011$$
$$4 \longrightarrow 0100 \qquad c \longrightarrow 1100$$
$$5 \longrightarrow 0101 \qquad d \longrightarrow 1101$$
$$6 \longrightarrow 0110 \qquad e \longrightarrow 1110$$
$$7 \longrightarrow 0111 \qquad f \longrightarrow 1111$$

$$0111_2 = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 7$$

Da esadecimale a binario

- Dato un numerale esadecimale per codificarlo in binario è sufficiente giustapporre le codifiche delle singole cifre (eliminando eventualmente zeri non significativi)
- Esempi:
  - $4c9f_{16}$ → 0100 1100 1001 1111 → 100110010011111_2
  - $b2a_{16}$ → 1011 0010 1010 → 101100101010_2
  - 157_{16} → 0001 0101 0111 → 101010111_2
- Nota bene che $157_{16} = 1 \cdot 16^2 + 5 \cdot 16^1 + 7 \cdot 16^0 = 343$

Da binario a esadecimale

- Per il passaggio di base da binario a esadecimale si effettua la codifica inversa avendo cura di aggiungere eventuali zeri non significativi in modo che la lunghezza del numerale in binario sia multipla di 4
- Esempi:
  - $10_2 \rightarrow 0010 \rightarrow 2_{16}$
  - $10011_2 \rightarrow 0001\ 0011 \rightarrow 13_{16}$
  - $111110010101100_2 \rightarrow 0111\ 1100\ 1010\ 1100 \rightarrow 7CAC_{16}$

Nota: comunemente un numerale esadecimale viene indicato con il prefisso 0x:

$7CAC_{16} \rightarrow 0x7CAC$

Rappresentazione e registri

- Un computer le grandezze numeriche sono elaborate mediante sequenze di simboli di lunghezza fissa dette parole
- Poichè una cifra esadecimale codifica 4 bit:
  - 1 byte (8 bit) → 2 cifre esadecimali
  - 4 byte (32 bit) → 8 cifre esadecimali
  - 8 byte (64 bit) → 16 cifre esadecimali

Esercizi

- Convertire da base 2 a base 10:
  - 0110011
  - 10101100
  - 1100110011

- Convertire in base 10 i seguenti numeri:
  - 102210_3
  - 431204_5
  - 5036_7
  - $198A1_{12}$

Esercizi

- Convertire da base 10 alla base indicata i seguenti numeri:
  - 7562 base 8
  - 1938 base 16
  - 205 base 16
  - 175 base 2

- In un registro a 32 bit è memorizzato il valore 0xF3A7C2A4. Esprimere il contenuto del registro in base 2
- Convertire da base 2 a base 16:
  - 10010
  - 11010101
  - 10010011
- Scrivere in babilonese il numero 4000

Numeri con parole di lunghezza fissa

- I registri dei moderni calcolatori sono tipicamente parole di 32 o 64 bit
- Con una parola sono rappresentabili un numero finito di naturali.
  - Nel caso di parole a 64 bit sono rappresentabili i numeri da 0 a 2^{64}-1
- Chiaramente, poiché ogni numero deve essere rappresentato dallo stesso numero di cifre, occorre ricorrere necessariamente a zeri non significativi

Numeri con parole di lunghezza fissa

- Supponiamo di avere una macchina che opera con parole di 16 bit (ancora usate in molte applicazioni embedded), il numero 9 sarà quindi rappresentato come: 0000 0000 0000 1001
- Operazioni come l'addizione o la moltiplicazione possono produrre numeri troppo grandi per essere rappresentati. In questo caso parleremo di trabocco o overflow
- Supponiamo di nuovo di avere una parola di 16 bit e supponiamo di voler elevare 1024 al quadrato. Questo produrrà un trabocco, infatti:

$$1024^2 = 1024 \cdot 1024 = 2^{10} \cdot 2^{10} = 2^{20} > 2^{16} - 1$$

Somma in binario

L'operazione di somma in binario è algoritmicamente analoga a quella decimale con la differenza che il riporto si ha quando si eccede 1 (invece che 9)

$$\begin{array}{r} \scriptstyle 11 \\ 4277 \\ +\ 5499 \\ \hline 9776 \end{array} \qquad \leftarrow \text{carries} \rightarrow \qquad \begin{array}{r} \scriptstyle 11 \\ 1011 \\ +\ 0011 \\ \hline 1110 \end{array}$$

(a) **(b)**

Overflow: al termine della somma ho un riporto di 1

$$\begin{array}{r} \scriptstyle 11\ \ 1\ \ \ \\ 1101 \\ +\ 0101 \\ \hline 10010 \end{array}$$

Moltiplicazione in binario

Anche la moltiplicazione in binario è analoga a quella decimale

$$
\begin{array}{ccccc}
0 & \times & 1 & = & 0 \\
1 & \times & 0 & = & 0 \\
0 & \times & 0 & = & 0 \\
1 & \times & 1 & = & 1
\end{array}
$$

```
    0101 ×
    0011 =
----------
    0101
   0101=
  0000==
 0000===
----------
 0001111
```

Sottrazione - prestito

$$\begin{array}{ccccc} 0 & - & 0 & = & 0 \\ 1 & - & 0 & = & 1 \\ 1 & - & 1 & = & 0 \\ \end{array}$$

$$\boxed{0 - 1 = 1}$$ Dopo il prestito dalla colonna a sinistra

$$\begin{array}{rccc} & & \scriptstyle 0 & \scriptstyle 10 \\ & 1 & \cancel{1} & \cancel{0} \\ - & 1 & 0 & 1 \\ \hline & 0 & 0 & 1 \end{array}$$

$$\begin{array}{rccccc} & & & \scriptstyle 1 & \scriptstyle 1 & \\ & & \scriptstyle 0 & \scriptstyle \cancel{10} & \scriptstyle \cancel{10} & \scriptstyle 10 \\ & 1 & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} \\ - & & & 1 & 1 & 1 \\ \hline & 1 & 0 & 0 & 0 & 1 \end{array}$$

$$\begin{array}{rccccc} & 1 & 0 & 0 & 0 & 1 \\ + & & & 1 & 1 & 1 \\ \hline & 1 & 1 & 0 & 0 & 0 \end{array}$$

- La sottrazione si effettua incolonnando i numeri
- Il prestito si propaga da destra verso sinistra

Rappresentazione di interi

- Occupiamoci ora della rappresentazione di numeri interi …, -2,-1,0,1,2,… mediante parole di lunghezza fissata
- Chiaramente dobbiamo codificarne non solo il valore assoluto ma anche il segno
- Le principali rappresentazioni di numeri interi sono:
  - Rappresentazione con segno
  - Complemento alla base (complemento a uno)
  - Complemento all’intervallo (complemento a due)

Rappresentazione con segno

- Si supponga che le parole siano di lunghezza m e la base sia b
- In tale rappresentazione la cifra più significativa di una parola rappresenta il segno. Per convenzione 0 rappresenta il segno più, 1 rappresenta il segno meno
- Le rimanenti m-1 cifre sono usate per rappresentare il valore assoluto di un intero
- Ad esempio si considerino parole binarie di lunghezza 4
  - 0100 → +100 → 4
  - 1011 → -011 →-3
  - Il più grande numero rappresentabile è 0111 (ovvero 7)
  - Il più piccolo numero rappresentabile è 1111 (ovvero -7)
  - Lo zero ha due possibili rappresentazioni: 0000 e 1000

Rappresentazione con segno

Poiché m-1 cifre sono utilizzate per rappresentare il valore assoluto, il range di numeri codificabili è $-(b^{m-1}-1),\ldots,0,\ldots,b^{m-1}-1$

Svantaggio: nelle operazioni di addizione o sottrazione occorre controllare il segno e i valori assoluti dei due operandi per determinare il segno del risultato.

- 0010 + 0001
  - sono entrambi positivi quindi il segno sarà positivo → 0011
- 0101 + 1010
  - il primo è positivo mentre il secondo è negativo, il valore assoluto del primo è maggiore del valore assoluto del secondo quindi il segno sarà positivo → 0011
- 1100+0011
  - il primo è negativo mentre il secondo è positivo, il valore assoluto del primo è maggiore di quello del secondo, quindi il segno sarà negativo → 1001
- 1011-1010
  - Entrambi sono negativi ma il valore assoluto del secondo è minore di quello del primo. Quindi occorre sottrarre 010 da 011 cambiando il bit del segno → 1001

Numeri negativi – Complemento a uno

8-bit ones'-complement integers

Bits	Unsigned value	Ones' complement value
0111 1111	127	127
0111 1110	126	126
0000 0010	2	2
0000 0001	1	1
0000 0000	0	0
1111 1111	255	−0
1111 1110	254	−1
1111 1101	253	−2
1000 0001	129	−126
1000 0000	128	−127

- Il complemento a 1 si ottiene complementando bit a bit (*bitwise*) tutte le cifre del numero binario
- Nell'aritmetica del complemento a 1, i numeri negativi sono rappresentati dal complemento del corrispondente numero positivo
- Con N bit si possono rappresentare i numeri compresi nell'intervallo $-(2^{N-1}-1)$, $2^{N-1}-1$
- Lo zero ha due rappresentazioni

Addizione – complemento a 1

- La somma avviene allineando le cifre.
- Il carry si propaga verso sinistra
- Se il carry supera l'ultimo digit, si ha una condizione di "end-around carry". Quando ciò accade, il bit va sommato al risultato parziale (esempio nelle prossime slide).

```
  0001 0110     22
+ 0000 0011      3
===========   ====
  0001 1001     25
```

Sottrazione in complemento a 1

- Nella sottrazione, il prestito (borrow) si propaga verso sinistra. Se il borrow supera l'ultimo digit, si genera una condizione che prende il nome di "**end-around borrow**". In questo caso, il bit deve essere sottratto al risultato paziale.

```
  0000 0110      6
- 0001 0011     19
===========    ====
1 1111 0011    -12    —An end-around borrow is produced, and the sign bit of the
intermediate result is 1.
- 0000 0001      1    —Subtract the end-around borrow from the result.
===========    ====
  1111 0010    -13    —The correct result (6 − 19 = -13)
```

Zero (positivo e negativo)

Adding negative zero:

carry

```
  0001 0110     22
+ 1111 1111     −0
===========   ====
1 0001 0101     21    An end-around carry is produced.
+ 0000 0001      1
===========   ====
  0001 0110     22    The correct result (22 + (−0) = 22)
```

Subtracting negative zero:

borrow

```
  0001 0110     22
− 1111 1111     −0
===========   ====
1 0001 0111     23    An end-around borrow is produced.
− 0000 0001      1
===========   ====
  0001 0110     22    The correct result (22 − (−0) = 22)
```

Zero negativo

```
  0001 0110     22
+ 1110 1001    −22
===========   ====
  1111 1111     −0    Negative zero.
```

Complemento alla base con N cifre

- Si definisce complemento alla base di un numero intero A in base b, rappresentato con N cifre, il numero b^N - A.
- Nel sistema di numerazione binario, considerando ad esempio 16 cifre, il complemento alla base di A è quindi 2^{16}–A
- Il complemento alla base di un numero binario con N cifre si può calcolare più semplicemente in questo modo:
  – si parte da destra (lsb) e si lasciano invariati gli zeri fino al primo 1;
  – il primo 1 resta invariato;
  – tutte le cifre successive vanno invertite, cioè gli 0 diventano 1 e gli 1 diventano 0.

Sottrazione – complemento alla base

- In questo metodo, si sfrutta la seguente uguaglianza:

$$B - A = B + (b^N - A) - b^N = B + \underline{A}_N - b^N$$

$$\text{dove} \quad \underline{A}_N = (b^N - A)$$

- la sottrazione B – A può essere eseguita sommando a B il complemento alla base di A = $\underline{A}_N$, trascurando poi il riporto che viene generato nella posizione N+1

- In binario, utilizzando n cifre per la rappresentazione, il riporto nella posizione N+1 viene eliminato automaticamente.

Complemento a 2

- La rappresentazione è analoga a quella unsigned con la differenza che il bit più significativo corrisponde a -2^{m-1} invece che 2^{m-1}

-2^{m-1}	2^{m-2}							2^1	2^0

- Lo zero ha una unica rappresentazione 0…0
- Il range di rappresentazione è $[-2^{m-1}, 2^{m-1} -1]$
- Essendo -2^{m-1} in valore assoluto il «peso» più grande, se un numero inizia con 1 allora è negativo altrimenti è positivo

Complemento a 2

- Minimo valore rappresentabile: $-2^{m-1} \rightarrow 10\ldots0$
- Massimo valore rappresentabile: $2^{m-1}-1 \rightarrow 01\ldots1$
- *Per complementare-alla-base un numero binario, si può anche invertire ogni cifra e poi sommare 1*
  - Rappresentare il numero -2 con 4 bit
  - $2 = 0010 \rightarrow -2 = (1101 + 0001) = 1110$
  - Attenzione! questo non vale per -2^{m-1} il cui complemento non è rappresentabile con m bit (i numeri positivi arrivano a $2^{m-1}-1$): $-2^{m-1} = 10\ldots0 \rightarrow (01\ldots1 + 0\ldots1) = 10\ldots0$
- *La somma avviene come per i numeri unsigned:*

  $-2+1 = 1110 + 0001 = 1111 = -1$

  $-7+7 = 1001 + 0111 = 0000 = 0$ (con riporto 1)

Overflow nel complemento a 2

- Sommare un numero negativo e uno positivo non genera overflow
- L'esempio -7+7 mostra come l'overflow non avviene come per gli unsigned quando ho riporto finale di 1
- L'overflow si verifica quando entrambi i numeri sono negativi (primo bit=1) o entrambi positivi (primo bit=0) e il risultato ha segno opposto

  4+5 = 0100+0101 = 1001 = -7

- Per estendere un numero in una rappresentazione con più bit basta riprodurre a sinistra il primo bit

  5=0101 → 00000101

  -4=1100 → 11111100

Carry e Overflow 1/3

Si supponga di lavorare con codifica complemento a 2 su 4 bit (-8,…,+7)
si consideri la seguente operazione di somma in complemento a due:

1001 (-7) e 1111 (-1).

La somma in base 10 è -8 e rappresenta ***il limite inferiore codificabile con 4 bit***
Eseguendo la somma tra le rappresentazioni dei numeri si ottiene:

```
   1001+
   1111=
 _________
 1  1000
```

❖l'operazione effettuata ha prodotto un risultato non contenibile nello spazio predisposto in quanto è necessario un altro bit (l'***1 a sinistra, ottenuto come riporto, viene*** memorizzato nel CARRY FLAG ***per superamento della capacità del registro),*** comunque il risultato ottenuto (1000 in complemento a 2 corrisponde a –8) è da considerare corretto.

Carry e Overflow 2/3

Si supponga sempre di lavorare con codifica in complemento a 2 su 4 bit (-8
Si consideri la seguente operazione di somma in complemento a due:

1001 (-7) e 1110 (-2).

In questo caso la somma -9 non è codificabile con 4 bit.
Eseguendo la somma si ottiene:

1001+
1110=
____
1 **0111**

Quando la somma di due interi produce come risultato un valore esterno all'insieme dei valori rappresentabili si dice che si è verificato un "Overflow" e il risultato ottenuto non è corretto;

Il Calcolatore non e in grado di *prevenire un errore di* Overflow, in quanto questo viene individuato solo dopo aver effettuato l'operazione.

- l'operazione effettuata ha prodotto un risultato non contenibile nello spazio predisposto in quanto è necessario un altro bit (l***'1 a sinistra, ottenuto come riporto, viene*** memorizzato nel CARRY FLAG ***per superamento della capacità del registro),*** ma il **risultato ottenuto (0111 - in complemento a 2 corrisponde a +7)) è da considerare errato!**

- Si ha Overflow.

Carry e Overflow 3/3

Si supponga di lavorare ad 8 bit (-128,….,+127)

Si consideri la seguente operazione di somma in complemento a due:

01111110 (126) e 00000011 (3).

In questo caso la somma (129) è superiore al numero massimo positivo codificabile in complemento a due. con 8 bit

Eseguendo la somma, si ottiene:

01111110 +
00000011 =
____
10000001

Quando la somma di due interi produce come risultato un valore esterno all'insieme dei valori rappresentabili si dice che si è verificato un "Overflow" e il risultato ottenuto non è corretto;

Il Calcolatore non e in grado di *prevenire un errore di* Overflow, in quanto questo viene individuato solo dopo aver effettuato l'operazione.

❖l'operazione effettuata ha prodotto un valore contenibile nello spazio predisposto ma **il risultato ottenuto (1**0000001 - **in complemento a 2 corrisponde a** -127) **) è da considerare errato!**

Generazione e test dell’Overflow

- Per capire se il risultato che è stato ottenuto sia valido o meno, osservando i casi precedenti basta controllare i bit più significativi dei numeri da sommare (X e Y) e della somma (S) ottenuta.
- Se i bit più significativi dei numeri da sommare (X e Y) sono diversi non potrà verificarsi mai l’overflow e il risultato sarà sempre da considerarsi corretto.
- Se i bit più significativi dei numeri da sommare (X e Y) sono uguali e il bit più significativo della somma (S) è diverso da essi allora ci sarà overflow e il risultato dovrà essere considerato errato.

Esempi 1/2

<u>Esempio:</u> Siano dati i numeri a 4 bit 0010 (+2) e 1010 (-6).

0010+
1010=
____
1100 (-4)

- **OF=0**, ossia il risultato S è valido, perché i bit più significativi di X e Y sono diversi.

Esempi 2/2

<u>Esempio:</u> Siano dati i numeri a 8 bit 01111110 (+126) e 00000011 (+3)

01111110+
00000011=
____
10000001

- **OF=1**, ossia il risultato S NON è valido, perché i bit più significativi di X e Y sono uguali e il bit più significativo di S NON è uguale a loro.

Esercizi

- Fornire la rappresentazione binaria in complemento a 2 ad 8 bit del numero -15
- Fornire la rappresentazione binaria in complemento a 2 ad 8 bit del numero -109
- Eseguire la somma dei numeri 5 e -32 espressa in complemento a 2 ad 8 bit
- Convertire in esadecimale il numero -53248 espresso in completamento a 2 ad 16 bit

Linguaggi e Numeri Interi: C

Tipo	Dichiarazione	bit	valori
Carattere con segno	`char`	8	$-128 \ldots 127$
Carattere senza segno	`unsigned char`	8	$0 \ldots 255$
Intero corto con segno	`short`	16	$-32768 \ldots 32767$
Intero corto senza segno	`unsigned short`	16	$0 \ldots 65535$
Intero con segno	`int`	32	$-2147483648 \ldots 2147483647$
Intero senza segno	`unsigned int`	32	$0 \ldots 4294967295$

Binary Coded Decimal

- L’elettronica degli elaboratori è binaria mentre la mente umana è abituata a ragionare in decimale.
  - I codici Binary Coded Decimal hanno lo scopo di fornire una naturale rappresentazione binaria del sistema numerico decimale.
  - Si parla di codici, al plurale, perché possono essere infiniti. Visto che possono essere infinite le codifiche binarie dei 10 simboli decimali (‘0’,…,’9’).
  - Quando si parla di codice BCD senza ulteriori specifiche si intende quello posizionale basato sulle potenze crescenti di due. Questo viene detto binario puro o 8421 in riferimento ai pesi di 4 bit letti dal più significativo (Most Significant Bit) al meno significativo (Least Significant Bit).
- Essendo 10 i simboli da codificare avremo bisogno di 4 bit.
- Notate che con 4 bit avremo 24=16 combinazioni che in binario puro corrispondono ai numeri 0,1,2,…9,10,…,15
- Poiché le combinazioni da 10 a 15 non si usano la codifica BCD è ridondante

Binary Coded Decimal

Cifra decimale	Cifra BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

i codici 1010, 1011, 1100, 1101, 1110 ,1111 non sono utilizzati (codifica ridondante)

Numeri decimali a più cifre di codificano giustapponendo le codifiche cifra per cifra

Esempio:

$23=0010\ 0011_{BCD}$

Nota che

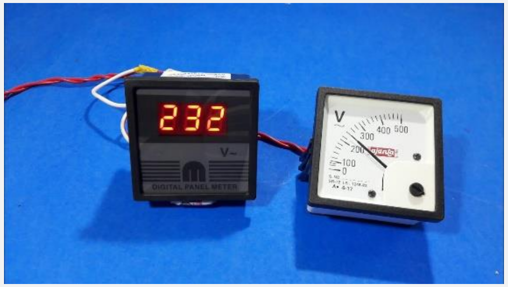
$23=00100011_{BCD} \neq 00100011_2=35$

Anche le somme differiscono:

$1000_{BCD}+0011_{BCD}=8+3=11=00010001_{BCD}$

e non 1011 (che è un codice non utilizzato)

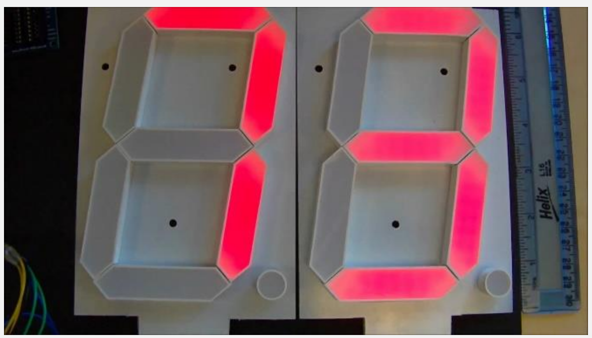
Binary Coded Decimal

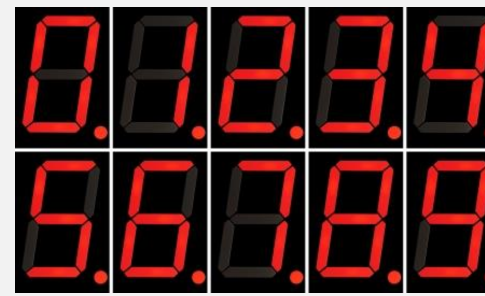


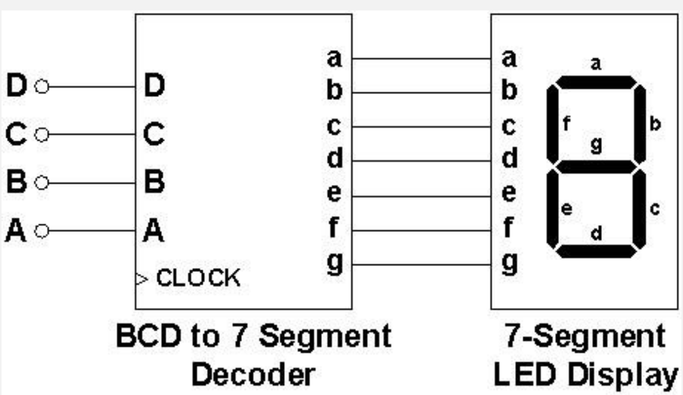
- La codifica BCD viene usata per presentare irisultati di una elaborazione numerica binaria.
- I quattro bit di ciascuna cifra codificata BCD vengono inviati ad un circuito di decodifica che provvederà a pilotare la cifra corrispondente.

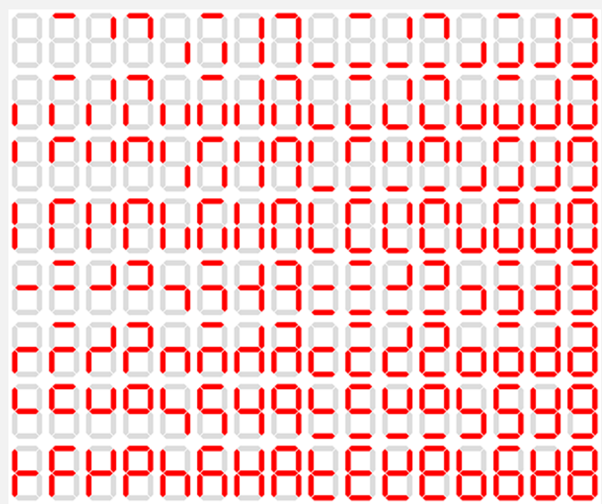
Display a 7 segmenti

- Il display è costituito da 7 diodi emettitori di luce (LED), modellati a forma di segmenti
- E’ possibile rappresentare simboli e caratteri alfanumerici, accendendo i segmenti opportuni

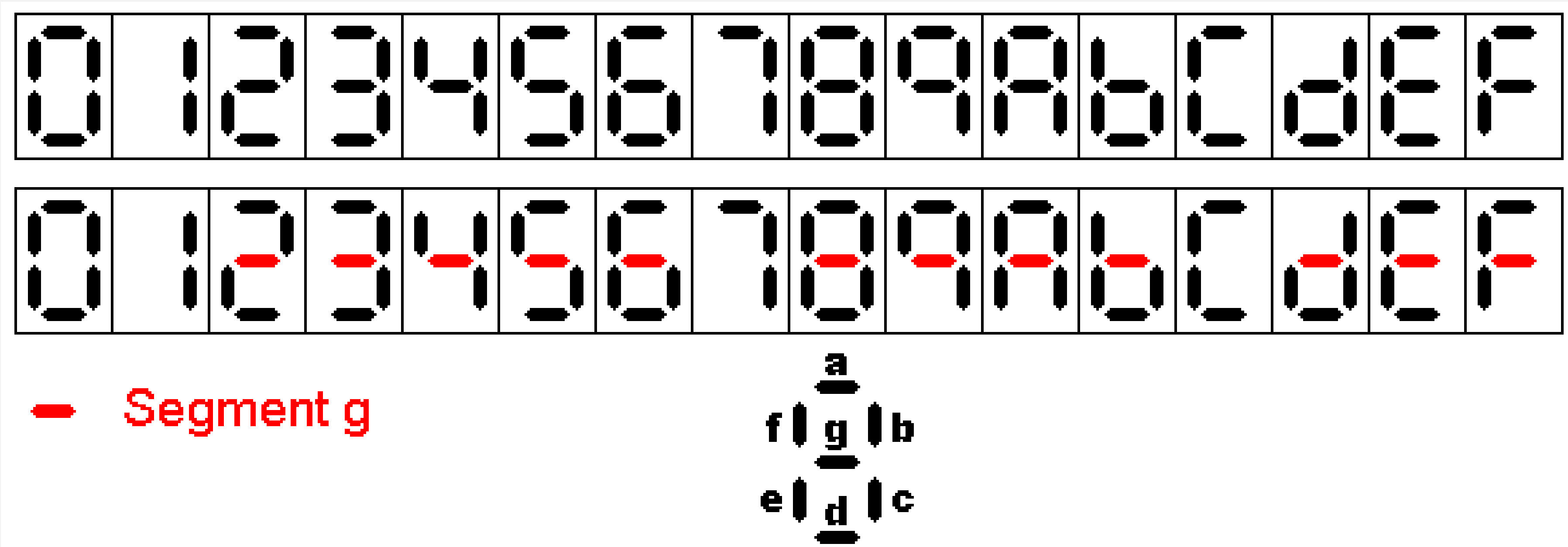








Simboli esadecimali



- E' possibile rappresentare tutti i simboli esadecimali, utilizzando la grafica in figura
- In questo modo, ogni cifra rappresenta un campo di 4 bit

Altri codici

Binario	Decimale	BCD	Eccesso-3	Biquinary	1 di 10
0	0	0000	0011	0100001	0000000001
1	1	0001	0100	0100010	0000000010
10	2	0010	0101	0100100	0000000100
11	3	0011	0110	0101000	0000001000
100	4	0100	0111	0110000	0000010000
101	5	0101	1000	1000001	0000100000
110	6	0110	1001	1000010	0001000000
111	7	0111	1010	1000100	0010000000
1000	8	1000	1011	1001000	0100000000
1001	9	1001	1100	1010000	1000000000

145 decimale

BCD
0001 0100 0101

Biquinary
0100010 0110000 1000001

Binario
10010001

Value	05-01234 bits
0	10-10000
1	10-01000
2	10-00100
3	10-00010
4	10-00001
5	01-10000
6	01-01000
7	01-00100
8	01-00010
9	01-00001

Bi-Quinary

Bi

Quinary

- Famiglia di codici, con diversi numeri di bit divisi in due campi (*bi* e *quinary*)
- Si contraddistinguono per avere un unico bit settato in ciascuno dei due campi
- Usato nell'IBM650 (dalla metà degli anni '50, fino al 1970)

Numeri con virgola

- Vediamo come rappresentare numeri razionali (con che risoluzione ?)
- Consideriamo un numero con virgola nella base naturale 10

$$c_{m-1}c_{m-2}\cdots c_0, c_{-1}\cdots c_{-k} = c_{m-1}\cdot 10^{m-1} + c_{m-2}\cdot 10^{m-2} + \cdots c_0 \cdot 10^0 + c_{-1}\cdot 10^{-1} + \cdots + c_{-k}\cdot 10^{-k}$$

- Per una generica base b abbiamo la generalizzazione

$$\sum_{i=-k}^{h-1} c_i \cdot b^i$$

Contributo dei bit

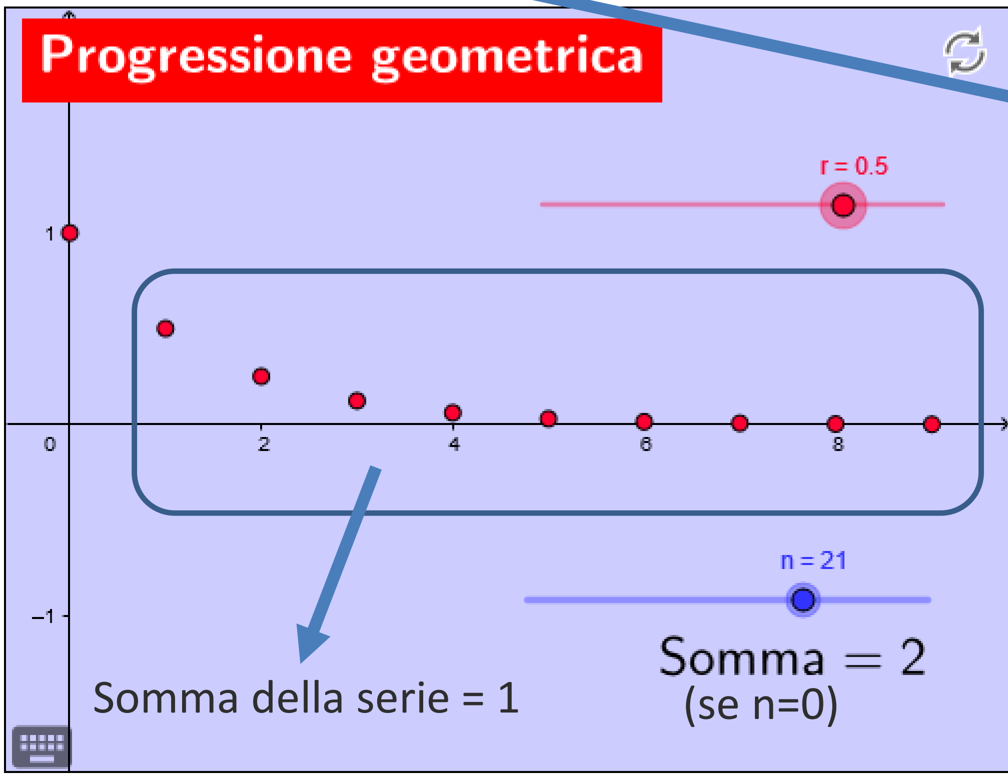
Table 1: Fixed-Point Examples

Binary	Hex	Integer	Floating Point Fraction	Fixed-Point Fraction	Actual
0100000000000000	4000	16384	0.50000000	0.50000000	1/2
0010000000000000	2000	8192	0.25000000	0.25000000	1/4
0001000000000000	1000	4096	0.12500000	0.12500000	1/8
0000100000000000	0800	2048	0.06250000	0.06250000	1/16
0000010000000000	0400	1024	0.03125000	0.03125000	1/32
0000001000000000	0200	512	0.01562500	0.01562500	1/64
0000000100000000	0100	256	0.00781250	0.00781250	1/128
0000000010000000	0080	128	0.00390625	0.00390625	1/256
0000000001000000	0040	64	0.00195312	0.00195312	1/512
0000000000100000	0020	32	0.00097656	0.00097656	1/1024
0000000000010000	0010	16	0.00048828	0.00048828	1/2048
0000000000001000	0008	8	0.00024414	0.00024414	1/4096
0000000000000100	0004	4	0.00012207	0.00012207	1/8192
0000000000000010	0002	2	0.00006104	0.00006104	1/16384
0000000000000001	0001	1	0.00003052	0.00003052	1/32768

Serie geometrica di ragione 1/2

$$\sum_{n=0}^{+\infty} q^n$$

- se il modulo della ragione q è minore di 1, ossia se $-1<q<1$, la serie geometrica converge ed ha per somma $\frac{1}{1-q}$



	A	B	C
1		TERMINI	SOMME PARZIALI
2	1	1	1
3	2	0.5	1.5
4	3	0.25	1.75
5	4	0.125	1.875
6	5	0.0625	1.9375
7	6	0.03125	1.96875
8	7	0.015625	1.984375
9	8	0.0078125	1.9921875
10	9	0.00390625	1.99609375
11	10	0.001953125	1.998046875
12	11	0.0009765625	1.9990234375
13	12	0.00048828125	1.99951171875
14	13	0.000244140625	1.999755859375
15			

Cambiamenti di base con virgola

- Per il cambiamento di un numero x da una base *a* ad una *b* si procede separatamente per la parte intera e per quella frazionaria
- Per la parte intera l’algoritmo chiaramente è quello visto in precedenza
- Per la parte frazionaria in procedimento è l’inverso:
  - si moltiplica la parte frazionaria di x per b (entrambi codificati in base a). La parte intera i del risultato sarà un numero da 0 a b-1 che, convertito in base b, costituisce la prima cifra di x in base b.
  - La parte frazionaria del risultato f si moltiplica ancora per b e la nuova parte intera, convertita in base b, costituisce la seconda cifra frazionaria.
  - Si procede iterativamente finché la parte frazionaria del risultato e’ zero o si e’ raggiunta la precisione desiderata

Cambiamenti di base con virgola

convertire in binario 0,625

$$0{,}625 \cdot 2 = 1,250 \quad i = 1 \quad f = 0,250 \quad c_{-1} = 1$$
$$0{,}250 \cdot 2 = 0,500 \quad i = 0 \quad f = 0,500 \quad c_{-2} = 0$$
$$0{,}500 \cdot 2 = 1,000 \quad i = 1 \quad f = 1,000 \quad c_{-3} = 1$$

$0{,}625 = 0{,}101_2$

Convertire $0{,}65_8$ in base 7 fino alla 3 cifra significativa

- Convertiamo prima in decimale

  $6 \times 8^{-1} + 5 \times 8^{-2} = 6 \times 0{,}125 + 5 \times 0{,}015625 = 0{,}75 + 0{,}078125 = 0{,}828125$

- Convertiamo 0,828125 in base 7: 0,554

  0,828125x7=5,796875 5

  0,796875x7=5,578125 5

  0,578125x7=4,046875 4

Rappresentazione in virgola fissa

- Ci occupiamo ora di rappresentare numeri non interi con parole (binarie) di lunghezza fissata m
- Nella rappresentazione in virgola fissa si suddividono gli m bit in due sottoparole (Qx.y)
  - i primi x bit (con $x<m$) sono dedicati alla codifica della parte intera
  - i rimanenti $y= m-x$ bit rappresentano la parte frazionaria
- Supponiamo di far uso di parole a 32 bit e di dedicare 20 bit per la parte intera e 12 per la parte frazionaria:
  - Massimo intero codificabile: $2^{19}-1$
  - Con 12 bit per la parte frazionaria si codificano circa 3 cifre decimali (ricordate $\log_2 10=3,32$)

Esempi

10011.011 Rappresentazione modulo e segno

- Segno: 1 *(1 = numero negativo; 0 = numero positivo)*
- Parte intera: $1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 = 1 + 2 = 3$
- Parte decimale: $0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 0.25 + 0.125 = 0.375$

Il numero ottenuto è: **-3.375**

11101.101
010111010.1101011

Risultato:

11101.101 => **-13.625**
010111010.1101011 => **186.8359375**

Esempi

Rappresentare in binario il numero **0.453125**:

0.453125 x 2 = 0.90625 => 0
0.90625 x 2 = 1.8125 => 1
0.8125 x 2 = 1.625 => 1
0.625 x 2 = 1.25 => 1
0.25 x 2 = 0.5 => 0
0.5 x 2 = 1.0 => 1

Il numero in binario è: **0.011101**

Convertire i seguenti numeri frazionari in binari:

0.15625
0.73543

Risultato:

0.15625 => 0.00101
0.73543 => 0.101111001...

Aritmetica in virgola fissa (complemento a due in Q3.3)

-1.25

-1.01_2

Compl. a 2

$1\,0\,.\,1\,1 \quad -1.25$

esteso a 3 bit

Segni opposti

$$
\begin{array}{rr}
1\,1\,0\,.\,1\,1 & -1.25 \\
+\ 0\,1\,1\,.\,0\,1\,0 & +3.25 \\
\hline
1\,0\,1\,0\,.\,0\,0\,0 & +2
\end{array}
$$

si ignora

Q3.3

$$
\begin{array}{rr}
1\,1\,1\,0\,.\,1\,1 & -1.25 \\
+\ 1\,1\,0\,0\,.\,0\,0\,1 & -3.875 \\
\hline
1\,0\,1\,0\,.\,1\,1\,1 & -5.125
\end{array}
$$

Q3.3

Minimo numero rappresentabile in Q3.3 (compl. a 2)

$100.000_2 = -4_{10}$

Risultato corretto in **Q4.3**

Risultato errato in **Q3.3**

Aritmetica in virgola fissa (complemento a due in Q4.3)

La stessa operazione in Q4.3:

esteso a 4.3

$$
\begin{array}{rrrr}
 & 1\,1\,1\,0\,.\,1\,1 & & -1.25 \\
+ & 1\,1\,0\,0\,.\,0\,0\,1 & & -3.875 \\
\hline
1 & 1\,0\,1\,0\,.\,1\,1\,1 & & -5.125
\end{array}
$$

Q4.3

si ignora

Segno corretto

Risultato corretto in Q4.3

Altri esempi

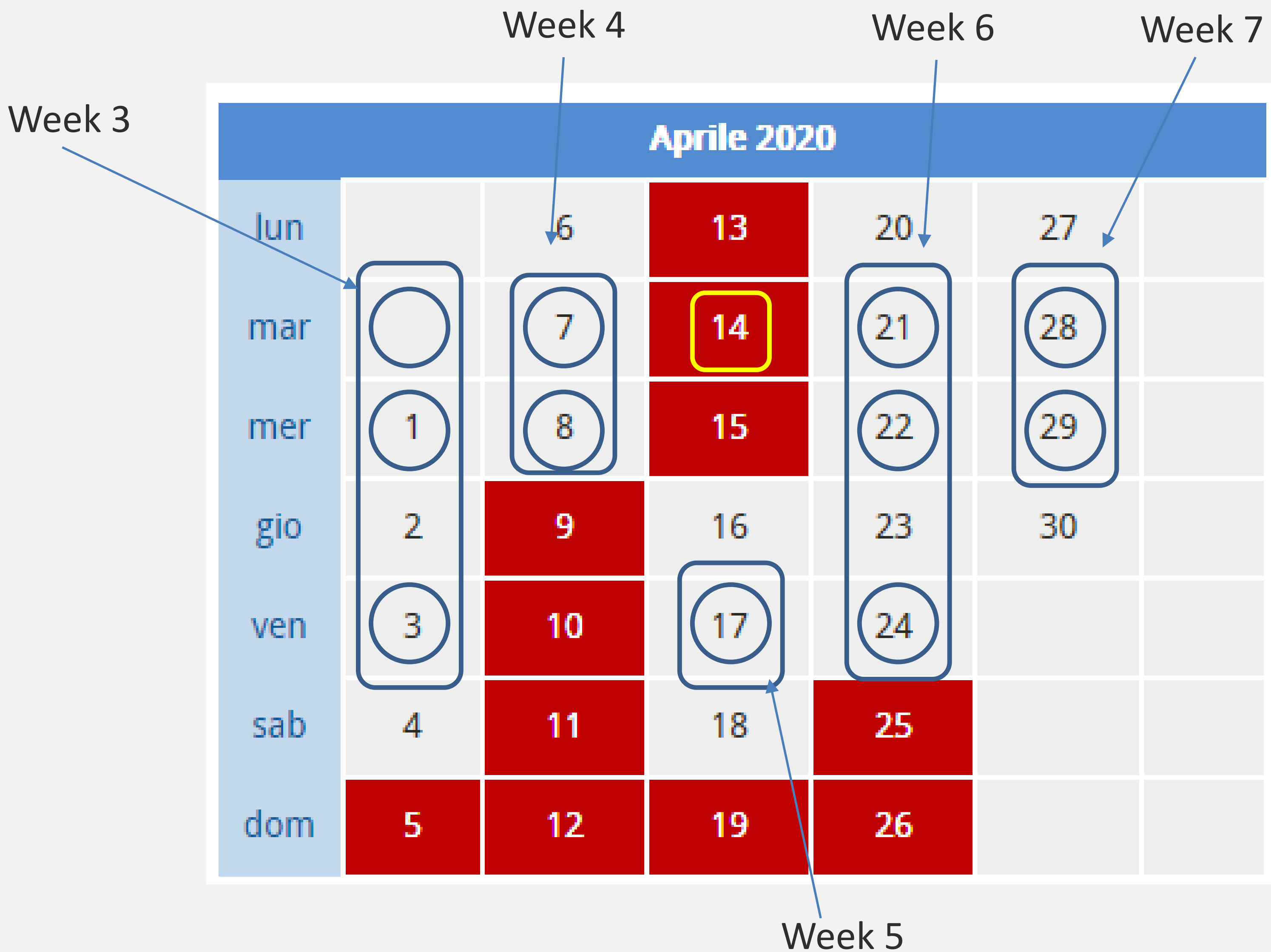
- Convertire i seguenti numeri in binario, con una precisione di 8 cifre decimali.

Decimale	Binario
23.466	*10111.01110111*
61.625	*111101.10100000*
13.543	*1101.10001011*
55.110	*110111.00011100*
19.999	*10011.11111111*
22.001	*10110.00000000*
41.700	*101001.10110011*

Esercizi

- Fornire la rappresentazione binaria in virgola fissa del numero -7.25 con 4 bit per la parte intera e 4 bit di parte frazionaria
- Fornire la rappresentazione binaria in codice esadecimale in virgola fissa del numero 2.33 con 4 bit per la parte intera e 4 bit di parte frazionaria, trascurando l'eventuale resto
- Fornire la rappresentazione binaria in virgola fissa del numero 55.4121 con 8 bit per la parte intera e 4 bit di parte frazionaria, trascurando l'eventuale resto
- Eseguire la somma 12.25+5.5 nella rappresentazione binaria in virgola fissa con 5 bit per la parte intera e 3 per quella frazionaria
- Eseguire la somma 9.875+10.5 nella rappresentazione binaria in virgola fissa con 5 bit per la parte intera e 3 per quella frazionaria

Calendario Accademico



Rappresentazione in virgola mobile

- Nella rappresentazione in virgola fissa disponendo di parole a 64 bit e supponendo di dedicarle tutte per la rappresentazione della parte frazionaria riusciremmo a codificare circa 18 cifre decimali
- Alcune applicazioni scientifiche operano con valori ancora più piccoli, altre invece con valori molto grandi
- In generale, fissare a priori la lunghezza della parte intera x e di quella frazionaria y costituisce una scelta rigida
- Per ovviare a queste difficoltà è stata introdotta una rappresentazione detta in virgola mobile

Rappresentazione in virgola mobile

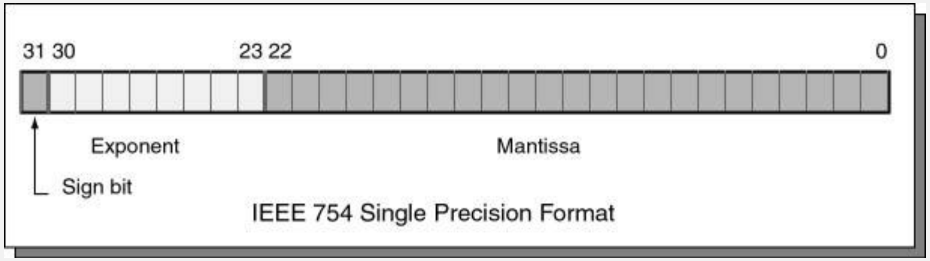
- Questa sfrutta la rappresentazione di un numero in una data base b:

$$x= (-1)^s \, m \, b^e$$

  - *s* determina il segno: 0 positivo, 1 negativo
  - *m* è detta mantissa
  - *e* è detto esponente
- Un numero reale è quindi rappresentato da una terna (s,m,e). Notate però che vi sarebbero infiniti modi di rappresentare x. Ad esempio, per 34,67 in base 10:
  - 3467 x 10-2
  - 3,467 x 10
  - 0,3467 x 102
- Useremo la rappresentazione scientifica in cui b > m ≥ 1
- In binario questo vuol dire che la mantissa è sempre del tipo 1,xyz
  - Chiaramente nella rappresentazione della mantissa non sprecherò memoria per rappresentare il primo bit "1" e lo considero sottointeso

Standard IEEE 754

- Una rappresentazione largamente adottata è quella dell' Institute of Electrical and Electronical Engineering (IEEE)
- Prevede 4 diversi formati per calcoli in singola o doppia precisione di tipo semplice o esteso (raramente usato)
- Descriveremo i tipi semplici
  - Singola precisione: 32 bit totali, 1 per il segno, 23 per la mantissa e 8 per l'esponente



IEEE 754 Single Precision Format

  - Doppia precisione: 64 bit totali, 1 per il segno, 52 per la mantissa e 11 per l'esponente

Esponente 1/2

- nello standard IEEE si usa la "rappresentazione polarizzata"
- In questa rappresentazione polarizzata, un valore **e** (compreso tra 0 e $2^k - 1$) codifica l'esponente **e**' = **e** – P, dove P = $2^{k-1} - 1$ (*polarizzazione*)
- Il più grande esponente rappresentabile è:

$$\mathbf{e_{max}}' = \mathbf{e_{max}} - P = (2^k - 1) - (2^{k-1} - 1) = 2^{k-1}$$

- Ad esempio sia k = 3 (P = $2^2 - 1 = 3$):

  e= 0 → e'=-3 e= 1 → e'=-2 e= 2 → e'=-1 e= 3 → e'=0
  e= 4 → e'=1 e= 5 → e'= 2 e= 6 → e'=3 e= 7 → e'=4

- Quindi ora gli esponenti sono codificati in ordine crescente da -3 (000_2) a 4 (111_2)

Esponente 2/2

- Esponente (8 bit)
  - Rappresentato in eccesso 127 (*polarizzazione* o *bias*)
  - L'intervallo di rappresentazione è [-127, 128]
  - Le due configurazioni estreme sono riservate, quindi

$$-126 \leq e \leq 127$$

  - Se gli 8 bit dell'esponente contengono $10100011 = 163_{10}$
    - L'esponente vale 163-127=36
  - Se gli 8 bit dell'esponente contengono $00100111 = 39_{10}$
    - L'esponente vale 39-127=-88

positivo

- Perché la polarizzazione?
  - Il numero più grande che può essere rappresentato è 11…11
  - Il numero più piccolo che può essere rappresentato è 00…00
  - Quindi, quando si confrontano due interi polarizzati, per determinare il minore basta considerarli come interi senza segno

negativo

Mantissa e rappresentazione dello zero

- Per la mantissa si adotta la rappresentazione scientifica 1,xyz…
- Riassumendo, ad (s, m, e) è associato il numero

$$x = (-1)^s \cdot 1{,}m \cdot 2^{e-P}$$

- Problema con questa rappresentazione non riesco a rappresentare lo 0.
- A questo scopo, lo standard IEEE considera delle eccezioni.

Dettagli dello Standard IEEE

Le combinazioni che corrispondono ad $m = 0$ ed $e = 255$ sono la rappresentazione di $\pm \infty$.

precisione singola

	$e=0$ (00000000_2)	$e=1,\ldots,254$	$e=255$ (11111111_2)
$m=0$	$(-1)^s \times 0$	$(-1)^s \times 1{,}0 \times 2^{e-127}$	$(-1)^s\ \infty$
$m \neq 0$	$(-1)^s \times \mathbf{0{,}m} \times 2^{-126}$	$(-1)^s \times 1{,}m \times 2^{e-127}$	NaN (indefinito)

0

Per $m = 0$ ed $e = 0$, si hanno due rappresentazioni dello 0, a seconda che s sia 1 (segno negativo) o 0 (segno positivo).

Le combinazioni che corrispondono ad $m \neq 0$ ed $e = 255$ sono la rappresentazione di forme indeterminate (Not-a-Number).

Per $m \neq 0$ ed $e = 0$ è prevista una rappresentazione per numeri molto vicini allo 0. Ricordate che m non è vincolato a iniziare con 1. Quindi se usassimo la rappresentazione usuale avrei, assumendo $s=0$, otterrei numeri maggiori di 2^{-127} : $1{,}m \times 2^{-127} > 2^{-127}$. Invece, per $m=00\ldots01$: $0{,}0\ldots01 \times 2^{-126} = 2^{-23} \times 2^{-126} = 2^{-149}$

Numeri Normalizzati

- Un numerale si intende in questa rappresentazione quando **e≠00000000**
- In questa rappresentazione, la mantissa è normalizzata tra 1 e 2: $\mathbf{1 \leq m < 2}$
- Quindi, la mantissa è sempre nella forma:

  1.XXXXXXXXX…X
- Si usano tutti i 23 bit per rappresentare la sola parte frazionaria (1 prima della virgola è implicito)
- Gli intervalli di numeri rappresentati sono pertanto:

  $(-2^{128},-2^{-126}]\quad[2^{-126},2^{128})$
  - Gli estremi sono esclusi perché il massimo valore assoluto di m è molto vicino a 2, ma è comunque inferiore
- L'intervallo $(-2^{-126}, 2^{-126})$ è detto *intervallo di underflow*

Numeri Denormalizzati

- Un numerale si intende in questa rappresentazione quando **e=00000000**
- L'esponente assume il valore *convenzionale* -126
- La mantissa è normalizzata tra 0 e 1: $\mathbf{0 < m < 1}$
- Quindi, la mantissa è sempre nella forma:

  0.XXXXXXXX…X
- Si usano tutti i 23 bit per rappresentare la sola parte frazionaria
- La più piccola mantissa vale 2^{-23}
- Gli intervalli rappresentati sono:

$$(-2^{-126},-2^{-149}] \quad [2^{-149},2^{-126})$$

NB *Più piccola è la mantissa, minore è il numero di cifre significative*

Estremi

- Più grande normalizzato: $\sim\pm 2^{128}$

  X 11111110 11111111111111111111111

  $\pm$ 2^{127} ~ 2

- Più piccolo normalizzato: $\pm 2^{-126}$

  X 00000001 00000000000000000000000

  $\pm$ 2^{-126} 1

- Più grande denormalizzato: $\sim \pm 2^{-126}$

  X 00000000 11111111111111111111111

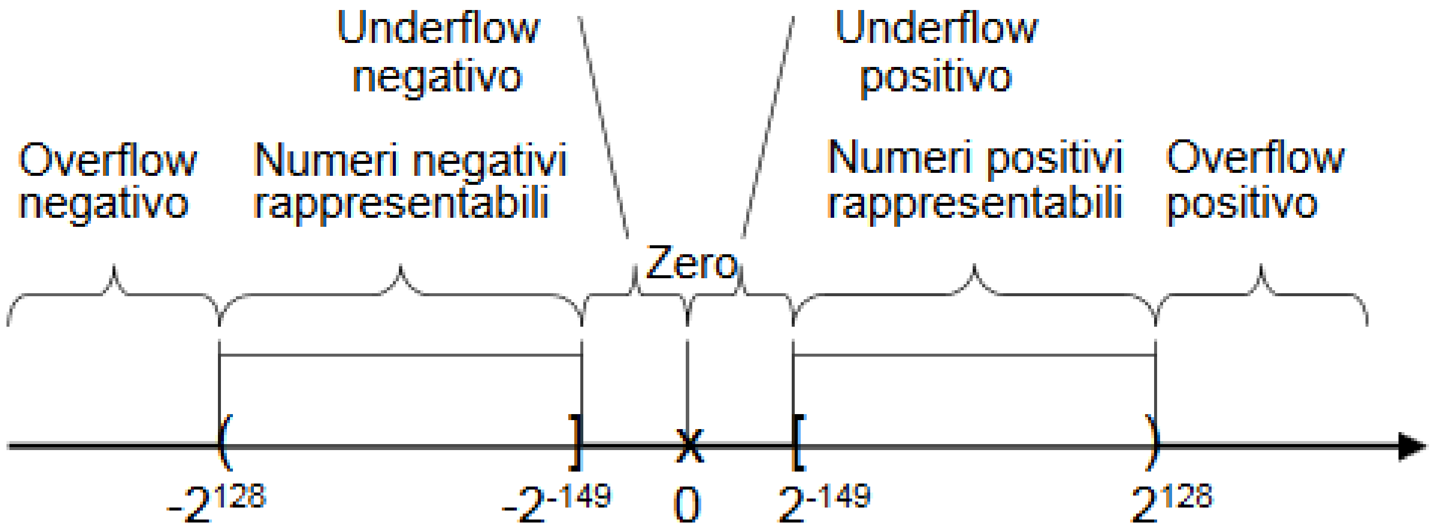
  $\pm$ 2^{-126} $(0.11\ldots)_2 \approx 1$

- Più piccolo denormalizzato: $\pm 2^{-149}$

  X 00000000 00000000000000000000001

  $\pm$ 2^{-126} $(0.00\ldots1)_2 = 2^{-23}$

Intervalli



- L'overflow può essere positivo

  Quando si devono rappresentare numeri positivi maggiori di 2^{128}
- L'overflow può essere negativo

  Quando si devono rappresentare numeri negativi minori di -2^{128}
- L'underflow può essere positivo

  Quando si devono rappresentare numeri positivi minori di 2^{-149}
- L'underflow può essere negativo

  Quando si devono rappresentare numeri negativi maggiori di -2^{-149}

Quadro Riassuntivo

Quadro riassuntivo

c \ m	0	$m \neq 0$
0	$v = 0$	$v = (-1)^S \times 2^{-126} \times 0.m$ NB
1 $1 \leq c \leq 254$ 254	$v = (-1)^S \times 2^{c-127} \times 1.m$	
255	$v=+\infty$ $s=0$ / $s=1$ $v=-\infty$	$v = \text{NaN}$

Rappresentazione di numeri molto piccoli in valore assoluto

La forma normalizzata (1 prima della virgola) impedisce rappresentazione valori v con $|v|<2^{-127}$

$\Rightarrow$ con c=0 si assume 0.m invece di 1.m così si possono usare anche i bit 0 della mantissa per "rimpicciolire"

Esempi f.p. in standard IEEE

- Il numero decimale 1021 è rappresentato dalla tripla (s, m, e):

  (0; 111 1111 0100 0000 0000 0000; 1000 1000)

  - s= 0 perché il numero è positivo.
  - La rappresentazione binaria di 1021 è:

    $1\ 111\ 1111\ 01 = (1,)\ 111\ 1111\ 01 \times 2^9$

  - m= 111 1111 0100 0000 0000 0000
  - Avendo 8 bit a disposizione per l'esponente, $P = 2^{8-1} - 1 = 2^7 - 1 = 127$.
  - L'esponente *e* si ricava invertendo la relazione di definizione della costante di polarizzazione:

$$e = e' + P = 9 + 127 = 136$$

che, convertito in binario, da 1000 1000

Esempio

Esempio di rappresentazione in precisione singola

$v = 42.6875_{10} = 101010.1011_2 = 1.010101011_2 \times 2^5$

Si ha

$s = 0$	(1 bit)
$c = 5 + K = 5_{10} + 127_{10} = 132_{10} = 10000100_2$	(8 bit)
$m = 01010101100000000000000$	(23 bit)

rappresentazione è giustapposizione di s, c, ed m: 0 10000100 01010101100000000000000

Altri Esempi

$$
\begin{aligned}
-30.375 &= (-11110.011)_{binario} \\
&= (-1.1110011)_{binario} \times 2^4 \\
&= (-1)1 \times (1 + 0.1110011) \times 2^{(131-127)}
\end{aligned}
$$

Ricordando che il formato IEEE 754 utilizza il seguente schema di rappresentazione

$(-1)segno \times (1 + significando) \times 2^{(esponente-127)}$

Abbiamo:

$segno = 1$

$esponente = 131 = (10000011)_{binario}$

$significando = (11100110000000000000000)_{binario}$

e quindi:

$(-30.375)10 = (1\ 10000011\ 11100110000000000000000)_{binario}$

Configurazione da convertire

`0 10001100 10001100000000000000000`

segno $0 \rightarrow$ segno $+$

esponente $10001100 \rightarrow 140$ decimale, a cui bisogna sottrarre la polarizzazione (127) per ottenere il vero esponente, cioè 13

significando $10001100000000000000000 \rightarrow 1 + 2^{-1} + 2^{-5} + 2^{-6} = 1,546875$

Pertanto il numero è dato da

$$+1 \times 1,546875 \times 2^{13} = 12672,0$$

Altri Esempi

- A quale valore corrisponde il seguente numero in virgola mobile in singola precisione?

1 10000100 00011011000000000000000

segno negativo:
-1

esponente:

$(10000100)_2$=132
L'esponente è rappresentato in eccesso 127, quindi esso vale effettivamente:
132 – 127 = **5**

significando:

$(\ ,00011011000......000)_2$ =
= $2^{-4}+2^{-5}+2^{-7}+2^{-8}$ = **0,10546875**
A parte i casi in cui l'esponente è composto da tutti **0** o da tutti **1**, il significando si intende preceduto dalla parte intera **1,** quindi la mantissa vale:
1 + 0,10546875 = **1,10546875**

$$(-1) * 2^5 * 1{,}10546875 = -35{,}375$$

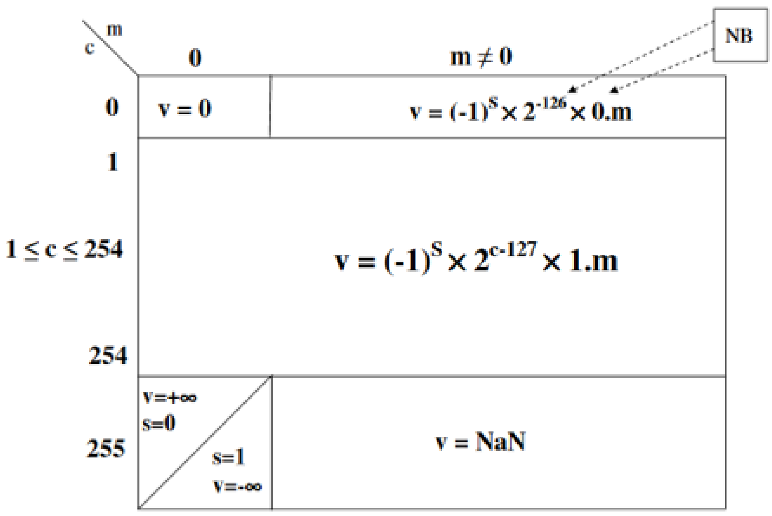
Altri Esempi

- A quale valore corrisponde il seguente numero in virgola mobile in singola precisione?

`11111111100011011000000000000000`

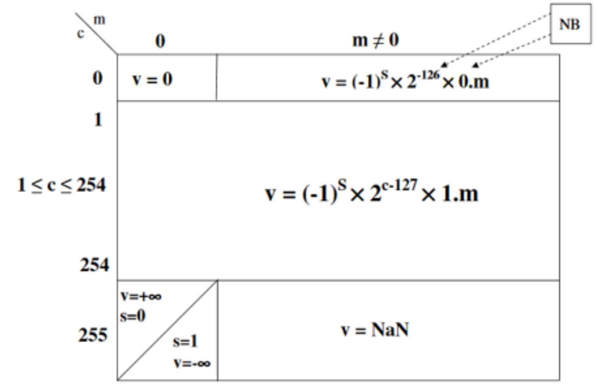
In questo caso, l'esponente contiene tutti `1`, ed inoltre il significando *non* è nullo:

la rappresentazione corrisponde al risultato di un'operazione non valida (ad es. 0/0) → **Not a Number (NaN)**



Altri Esempi

- A quale valore corrisponde il seguente numero in virgola mobile in singola precisione?



1 00000000 00011011000000000000000

segno negativo: **-1**

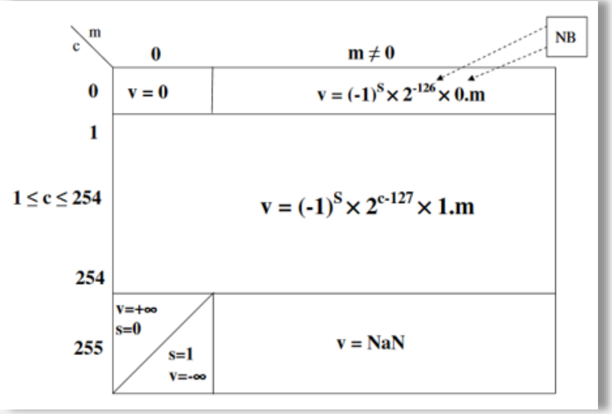
esponente:
La sequenza (00000000) corrisponde alla rappresentazione denormalizzata.
L'esponente è in questo caso **-126** (**non -127 !**)

significando:
(,00011011000......000$)_2$ =
= $2^{-4}+2^{-5}+2^{-7}+2^{-8}$ = **0,10546875**
Nella rappresentazione <u>denormalizzata</u> il significando si intende preceduto dalla parte intera **0,** quindi la mantissa qui vale:
0 + 0,10546875 = **0,10546875**

(-1) * 2^{-126} * 0,10546875

<u>Osservazione</u>: Il numero ha *meno cifre significative* rispetto alla rappresentazione normalizzata (poiché la parte intera è nulla, gli **0** a sinistra nella mantissa non sono significativi, sono cioè cifre "inutilizzate")

Altri Esempi



A quale valore corrisponde il seguente numero in virgola mobile in singola precisione?

`11111111100000000000000000000000`

In questo caso, l'esponente contiene tutti `1`, ed inoltre il significando è nullo:

la rappresentazione corrisponde ad *inifinito*, precisamente $-\infty$, ottenuto ad esempio da un'operazione come $-5 / 0$

Altri Esempi

c \ m	0	m ≠ 0
0	v = 0	$v = (-1)^S \times 2^{-126} \times 0.m$
1 ≤ c ≤ 254	$v = (-1)^S \times 2^{c-127} \times 1.m$	
255	v=+∞ s=0; s=1 v=-∞	v = NaN

NB

- Convertire in virgola mobile in singola precisione il valore decimale **+19,4375**

- Numero positivo → bit di segno 0
- Adesso, riscriviamo il valore assoluto del numero nella forma $2^e * 1,xxxxxx$
  – essendo il numero maggiore di, o uguale a 2, procediamo per divisioni successive:
  $19,4375 = 2^1 * 9,71875 = 2^2 * 4,859375 =$
  $= 2^3 * 2,4296875 = 2^4 *$ **1,21484375**
- L'esponente è 4, che in eccesso 127 è rappresentato come:
  4+127 = 132 → 10000011
- La mantissa è 1,21484375, quindi il significando è 0,21484375
  – dobbiamo rappresentare questo valore in virgola fissa su 23 bit con il procedimento delle moltiplicazioni successive

Altri Esempi

c \ m	0	$m \neq 0$
0	$v = 0$	$v = (-1)^S \times 2^{-126} \times 0.m$
$1 \leq c \leq 254$ (1 … 254)		$v = (-1)^S \times 2^{c-127} \times 1.m$
255	$v=+\infty$, $s=0$ / $s=1$, $v=-\infty$	$v = NaN$

NB

0,21484375

– dobbiamo rappresentare questo valore in virgola fissa su 23 bit con il procedimento delle moltiplicazioni successive

0,21484375	* 2 =	0, 4296875
0,4296875	* 2 =	0, 859375
0,859375	* 2 =	1, 71875
0,71875	* 2 =	1, 4375
0,4375	* 2 =	0, 875
0,875	* 2 =	1, 75
0,75	* 2 =	1, 5
0,5	* 2 =	1, 0
0,0	* 2 =	0, 0

- In virgola mobile in singola precisione il valore decimale **+19,4375** è rappresentato come

0 10000011 00110111000000000000000

Altri Esempi

- Convertire in virgola mobile in singola precisione il valore decimale **- 0,1796875**

- Numero negativo → bit di segno 1
- Adesso, riscriviamo il valore assoluto del numero nella forma $2^e * 1{,}xxxxxx$
  - essendo il numero minore di 1, procediamo per moltiplicazioni successive:

    $0{,}1796875 = 2^{-1} * 0{,}359375 = 2^{-2} * 0{,}71875 =$
    $= 2^{-3} * \mathbf{1{,}4375}$

- L'esponente è -3, che in eccesso 127 è rappresentato come:

  -3+127 = 124 → 01111100
- La mantissa è 1,4375, quindi il significando è 0,4375
  - dobbiamo rappresentare questo valore in virgola fissa su 23 bit con il procedimento delle moltiplicazioni successive

0, 4375	* 2 =	0, 875
0,875	* 2 =	1, 75
0,75	* 2 =	1, 5
0,5	* 2 =	1, 0
0,0	* 2 =	0, 0
0,0	* 2 =	0, 0

- In virgola mobile in singola precisione il valore decimale **- 0,1796875** è rappresentato come

  1 01111100 01110000000000000000000

Altri Esempi

- Convertire in virgola mobile in singola precisione il valore decimale **- 2,6**

- Numero negativo → bit di segno 1
- Adesso, riscriviamo il valore assoluto del numero nella forma $2^e * 1{,}xxxxxx$
  - essendo il numero maggiore di, o uguale a 2, procediamo per divisioni successive:

    $2{,}6 = 2^{1} * \mathbf{1{,}3}$

- L'esponente è 1, che in eccesso 127 è rappresentato come:

  1+127 = 128 → 10000000
- La mantissa è 1,3, quindi il significando è 0,3
  - dobbiamo rappresentare questo valore in virgola fissa su 23 bit con il procedimento delle moltiplicazioni successive

0,3	* 2 =	0, 6
0,6	* 2 =	1, 2
0,2	* 2 =	0, 4
0,4	* 2 =	0, 8
0,8	* 2 =	1, 6
0,6	* 2 =	1, 2
0,2	* 2 =	0, 4
0,4	* 2 =	0, 8
0,8	* 2 =	1, 6
…	…	…

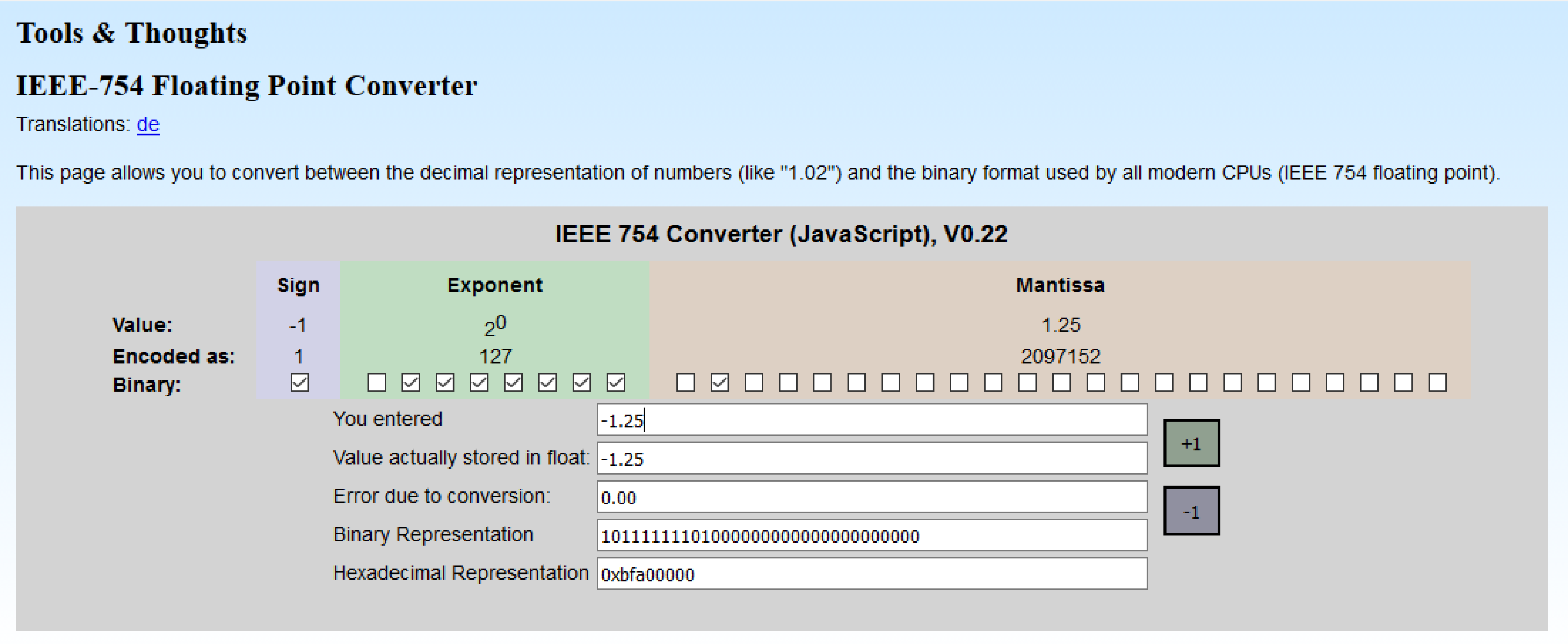
<u>Osservazione</u>:
il numero è periodico in base 2
Non è possibile una rappresentazione esatta!

Esercizi

- Rappresentare nel formato IEEE 754
  - 1.25
  - 10
  - -0.59375
  - 1007
  - 3.875

https://www.h-schmidt.net/FloatConverter/IEEE754.html

-1.25



https://www.h-schmidt.net/FloatConverter/IEEE754.html

-10

Tools & Thoughts

IEEE-754 Floating Point Converter

Translations: [de]

This page allows you to convert between the decimal representation of numbers (like "1.02") and the binary format used by all modern CPUs (IEEE 754 floating point).

IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	-1	2^3	1.25
Encoded as:	1	130	2097152
Binary:	☑	☑ ☐ ☐ ☐ ☐ ☐ ☑ ☐	☐ ☑ ☐

You entered	-10
Value actually stored in float:	-10
Error due to conversion:	0
Binary Representation	11000001001000000000000000000000
Hexadecimal Representation	0xc1200000

+1

-1

https://www.h-schmidt.net/FloatConverter/IEEE754.html

-0.59375

Tools & Thoughts

IEEE-754 Floating Point Converter

Translations: de

This page allows you to convert between the decimal representation of numbers (like "1.02") and the binary format used by all modern CPUs (IEEE 754 floating point).

IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	-1	2^{-1}	1.1875
Encoded as:	1	126	1572864
Binary:	☑	☐☑☑☑☑☑☑☐	☐☐☑☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

You entered	-0.59375
Value actually stored in float:	-0.59375
Error due to conversion:	0.00000
Binary Representation	10111111000110000000000000000000
Hexadecimal Representation	0xbf180000

+1

-1

https://www.h-schmidt.net/FloatConverter/IEEE754.html

1007

Tools & Thoughts

IEEE-754 Floating Point Converter

Translations: de

This page allows you to convert between the decimal representation of numbers (like "1.02") and the binary format used by all modern CPUs (IEEE 754 floating point).

IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	+1	2^9	1.966796875
Encoded as:	0	136	8110080
Binary:	☐	☑ ☐ ☐ ☐ ☑ ☐ ☐ ☐	☑ ☑ ☑ ☑ ☐ ☑ ☑ ☑ ☑ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

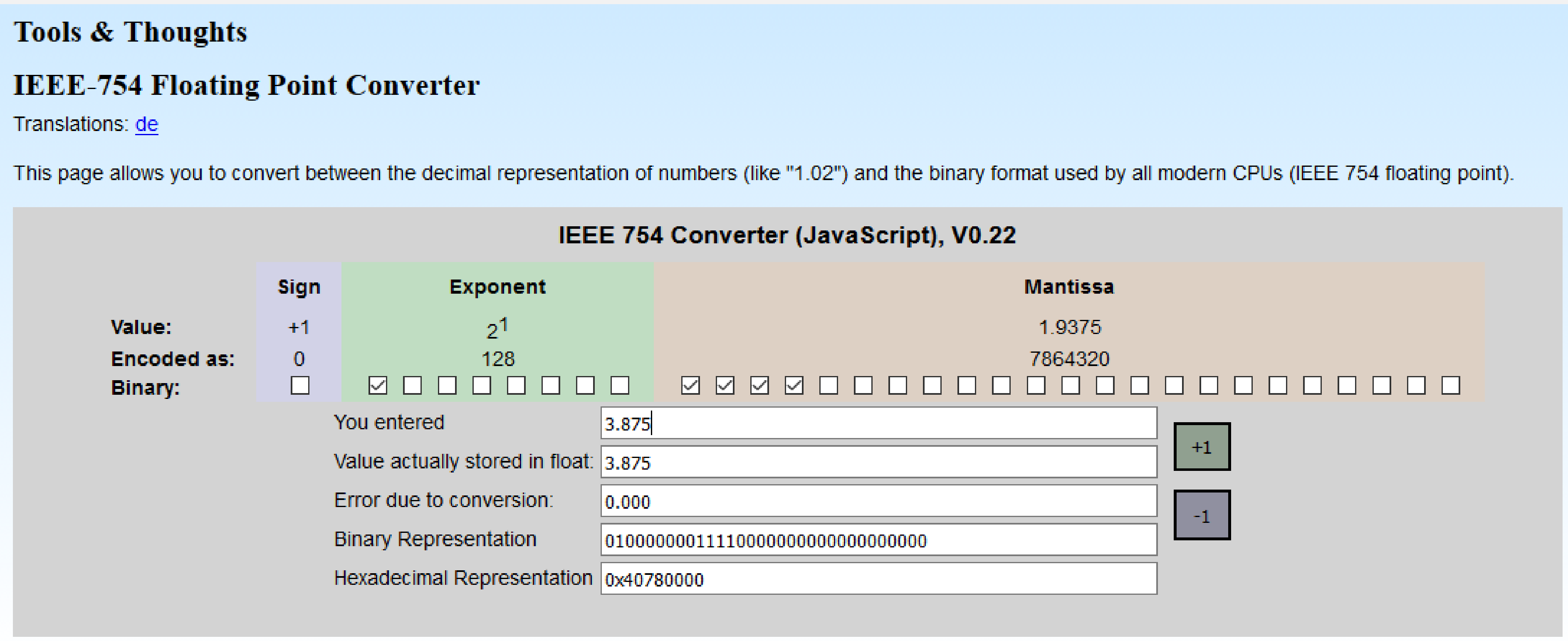
You entered	1007
Value actually stored in float:	1007
Error due to conversion:	0
Binary Representation	01000100011110111100000000000000
Hexadecimal Representation	0x447bc000

+1

-1

https://www.h-schmidt.net/FloatConverter/IEEE754.html

3.875



https://www.h-schmidt.net/FloatConverter/IEEE754.html

Gaps

- La *gap* è la minima distanza tra due numeri rappresentabili

Actual Exponent (unbiased)	Exp (biased)	Minimum	Maximum	Gap
−1	126	0.5	≈ 0.999999940395	≈ 5.96046e-8
0	127	1	≈ 1.999999880791	≈ 1.19209e-7
1	128	2	≈ 3.999999761581	≈ 2.38419e-7
2	129	4	≈ 7.999999523163	≈ 4.76837e-7
10	137	1024	≈ 2047.999877930	≈ 1.22070e-4
11	138	2048	≈ 4095.999755859	≈ 2.44141e-4
23	150	8388608	16777215	1
24	151	16777216	33554430	2
127	254	≈ 1.70141e38	≈ 3.40282e38	≈ 2.02824e31

interi

$$\left\{\begin{array}{l} 2^{23} \text{ x } 1{,}00..0 = 2^{23} \\ 2^{23} \text{ x } 1{,}11..1 = 2^{24} - 1 \end{array}\right.$$

$$\left\{\begin{array}{l} 2^{24} \text{ x } 1{,}00..0 = 2^{24} \\ 2^{24} \text{ x } 1{,}11..1 = 2^{25} - 1 \end{array}\right.$$

interi pari

	Sign	Exponent	Mantissa
Value:	+1	2^{23}	1.9999998807907104
Encoded as:	0	150	8388607
Binary:	☐	☑ ☐ ☐ ☑ ☐ ☑ ☑ ☐	☑ ☑

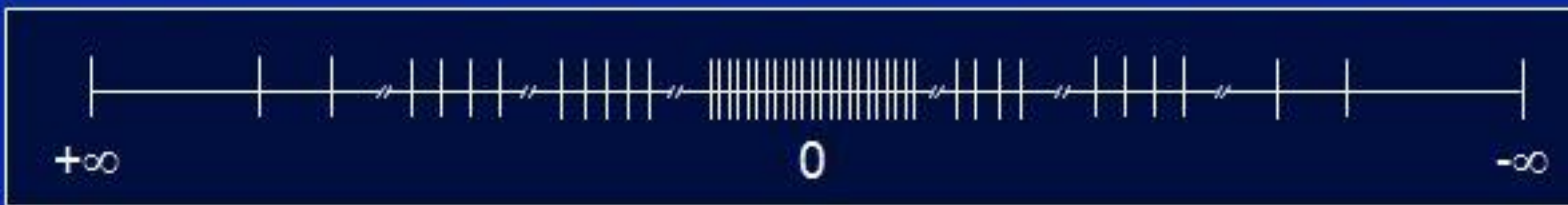
You entered	16777215
Value actually stored in float:	16777215
Error due to conversion:	0
Binary Representation	01001011011111111111111111111111
Hexadecimal Representation	0x4b7fffff

+1

-1

Gaps

Floating-Point: non-uniform distribution (variable precision)



IQ Fractions: uniform distribution (same precision everywhere)

- Both floating-point and IQ formats have 2^{32} possible values on the number line
- It's how each distributes these values that differs

Altre conversioni

-378.125 = 1 - 1000 0111 - 011 1101 0001 0000 0000 0000

-375.375 = 1 - 1000 0111 - 011 1011 1011 0000 0000 0000

10 011 = 0 - 1000 1100 - 001 1100 0110 1100 0000 0000

-374 = 1 - 1000 0111 - 011 1011 0000 0000 0000 0000

-370 = 1 - 1000 0111 - 011 1001 0000 0000 0000 0000

-37.1 = 1 - 1000 0100 - 001 0100 0110 0110 0110 0110

-366.625 = 1 - 1000 0111 - 011 0111 0101 0000 0000 0000

-362.651 5 = 1 - 1000 0111 - 011 0101 0101 0011 0110 0100

-361.687 5 = 1 - 1000 0111 - 011 0100 1101 1000 0000 0000

-360.76 = 1 - 1000 0111 - 011 0100 0110 0001 0100 0111

-359.5 = 1 - 1000 0111 - 011 0011 1100 0000 0000 0000

-357.33 = 1 - 1000 0111 - 011 0010 1010 1010 0011 1101

-356 = 1 - 1000 0111 - 011 0010 0000 0000 0000 0000

Example Converting from IEEE 754 Form

Suppose we wish to convert the following single-precision IEEE 754 number into a floating-point decimal value:

11000000110110011001100110011010

1. *First, we divide the bits into three groups:*

   1 10000001 10110011001100110011010

   The first bit shows us the sign of the the number.
   The next 8 bits give us the exponent.
   The last 23 bits give us the fraction.

2. *Now we look at the sign bit*

   If this bit is a 1, the number is negative; if it is 0, the number is positive. Here, the bit is a 1, so the number is negative.

3. *Next, we get the exponent and the correct bias*

   To get the exponent, we simply convert the binary number 10000001 back to base-10 form, yielding 129

   Remember that we will have to subtract an appropriate bias from this exponent to find the power of 2 we need. Since this is a single-precision number, the bias is 127.

4. *Then we must convert the fraction bits back into base 10*

   To do this, we multiply each digit by the corresponding power of 2 and sum the results:

$$\begin{aligned} 0.10110011001100110011010_{\text{binary}} &= 1\cdot 2^{-1} + 0\cdot 2^{-2} + 1\cdot 2^{-3} + 1\cdot 2^{-4} + 0\cdot 2^{-5} + \cdots \\ &= 1/2 + 1/8 + 1/16 + \cdots \\ &= 0.7000000476837158 \end{aligned}$$

   Remember, this number is most likely just an approximation of some other number. There will most likely be some error.

5. *We have all the information we need. Now we just calculate the following expression:*

$$\begin{aligned} (-1)^{\text{sign bit}}(1 + \text{fraction}) \times 2^{\text{exponent - bias}} &= (-1)^1(1.7000000476837158) \times 2^{129-127} \\ &= -6.800000190734863 \end{aligned}$$

Costanti... Fino a prova contraria...

Grandezza	Simbolo usuale	Valore	unità	legge fisica
Velocità della luce nel vuoto	c	299 792 458	$m \cdot s^{-1}$	Equaz.di Maxwell
Costante dielettrica del vuoto	ε_0	$8{,}854\ 187\ 817... \times 10^{-12}$	$F \cdot m^{-1}$	Equaz.di Maxwell
Permeabilità del vuoto	μ_0	$4\pi \times 10^{-7}$	$T \cdot m \cdot A^{-1}$	Equaz.di Maxwell
Costante di gravitazione universale	G	$6{,}672\ 59(85) \times 10^{-11}$	$N \cdot m^2 \cdot kg^{-2}$	Legge di gravitazione
Costante di Planck	h	$6{,}626\ 068\ 76(52) \times 10^{-34}$	$J \cdot s$	Effetto fotoelettrico
Carica dell'elettrone	e	$1{,}602\ 176\ 462(63) \times 10^{-19}$	C	
Massa a riposo dell'elettrone	m_e	$9{,}109\ 381\ 88(72) \times 10^{-31}$	kg	
Massa a riposo del protone	m_p	$1{,}672\ 621\ 58(13) \times 10^{-27}$	kg	
Massa a riposo del neutrone	m_n	$1{,}674\ 927\ 16(13) \times 10^{-27}$	kg	
Unità di massa atomica	1 amu	$1{,}660\ 538\ 73(13) \times 10^{-27}$	kg	
Numero di Avogadro	L oppure N_A	$6{,}022\ 141\ 99(47) \times 10^{23}$	mol^{-1}	
Costante di Boltzmann	k	$1{,}380\ 6503(24) \times 10^{-23}$	$J \cdot K^{-1}$	Legge dei gas
Costante di Faraday	F	$9{,}648\ 534\ 15(39) \times 10^4$	$C \cdot mol^{-1}$	
Costante dei gas	R	8,314 472(15)	$J \cdot K^{-1} \cdot mol^{-1}$	
Costante di struttura fine	α	$7{,}297\ 352\ 533(27) \times 10^{-3}$		
Raggio di Bohr	a_0	$5{,}291\ 772\ 083(19) \times 10^{-11}$	m	
Costante di Rydberg	R_∞	$1{,}097\ 373\ 156\ 8549(83) \times 10^7$	m^{-1}	

Confronti 1/2

- Per stabilire quale di due numeri in virgola mobile sia il maggiore
- Se sono di segno *discorde*, allora il numero positivo è maggiore
- Se sono di segno *concorde*
  – Se sono *positivi*
    - Il numero con l'esponente *più grande* è il maggiore; a parità di esponente, il numero con mantissa *più grande* è maggiore
  – Se sono *negativi*
    - Il numero con l'esponente *più piccolo* è il maggiore; a parità di esponente, il numero con mantissa *più piccola* è maggiore

Confronti 2/2

- Per due numeri positivi (negativi)
  - Siano a e b due numeri positivi (negativi) rappresentati in virgola mobile da $a_{31}a_{30}\ldots a_0$ e $b_{31}b_{30}\ldots b_0$
  - Notare che $a_{31} = b_{31} = 0$ (1)
- Per verificare quale dei due sia il maggiore, non occorre nessuna conversione
  - È sufficiente confrontarli come se fossero interi senza segno
  - Basta scorrere i bit, ed al primo bit diverso si individua il maggiore
    - Il numero con l'i-esimo bit a 1 (0)
    - Il numero con esponente/mantissa più grande (più piccolo) è il maggiore

Operazioni in virgola mobile

- La moltiplicazione fra due numeri n1 ed n2 rappresentati in virgola mobile dalle triple (s1, m1 , e1) ed (s2, m2 , e2) ha per risultato il numero rappresentato dalla tripla: (s , e , m) in cui:
  - s = 0 se s1 = s2 oppure: s = 1 se s1 ≠ s2
  - e = e1 + e2
  - m = m1 x m2
  - Dopo l'operazione di solito è necessaria la normalizzazione del risultato
- La divisione si effettua con regole analoghe.
- L'addizione e la sottrazione sono più complesse perché prima di effettuarle bisogna rendere uguali gli esponenti.
  - Durante questa operazione se i numeri sono uno molto grande ed uno molto piccolo, per effetto dello scorrimento delle mantisse per pareggiare gli esponenti, si possono perdere cifre significative.

Moltiplicazione

- Si moltiplicano le mantisse e si sommano algebricamente gli esponenti
- Se necessario si scala la mantissa per normalizzarla e si riaggiusta l'esponente
- Esempio (*Notazione IEEE 754*)

$n_3 = n_1 \times n_2$

n_1 : 0 10011001 10010111011100101100111

n_2 : 1 10101010 10000000000000000000000

$e_1 = (26)_{10}$, $e_2 = (43)_{10}$

- $e_1 + e_2 = (69)_{10} = 11000100$
- $m_1 \times m_2 = 10.0110001100101100001101 0$

N.B.: ricordare il bit implicito delle mantisse nella moltiplicazione

- si scala la mantissa di un posto
- si aumenta di 1 l'esponente

n_3 : 1 11000101 00110001100101100001101

Perdita di cifre significative

- Vediamo con un esempio perché si possono perdere cifre significative effettuando una somma. Per semplicità, considereremo una coppia di numeri decimali espressi attraverso una mantissa di 4 cifre ed un esponente di una sola cifra:

$$n1 = 0,3435 \times 10^3 \text{ ed } n2 = 0,9970 \times 10^5.$$

- Per effettuare la somma si può portare l'esponente di n1 da 3 a 5. Siccome ciò equivale a moltiplicare di fattore 100, per mantenere il valore costante occorre contemporaneamente dividere per 100 la mantissa:

$$0,3435 \times 10^3 = 0,\ 003435 \times 10^5$$

- Poiché le cifre della mantissa sono 4, occorre effettuare un arrotondamento. Per difetto otterremmo 0,0034.
- La somma dei due numeri risulta: $1,0004 \times 10^5$, che dopo la normalizzazione diventa $0,1000 \times 10^6$

Somma

- Per addizionare e sottrarre occorre scalare le mantisse per eguagliare gli esponenti
- Esempio (*Notazione IEEE 754*)

$n_1 + n_2$

n_1 : 0 10011001 00010111011100101100111

n_2 : 0 10101010 11001100111000111000100

$e_1 = (26)_{10}$, $e_2 = (43)_{10}$:

– occorre scalare m_1 di 17 posti

Bit implicito della mantissa

n'_1 : 0 10101010 00000000000000001000101 +

n_2 : 0 10101010 11001100111000111000100

0 10101010 11001100111001000001001 Somma delle mantisse

0 10101010 11001100111001000001010 Round

- Notare che l'addendo più piccolo perde cifre significative

Accuratezza

- L'aritmetica FP, a causa dei limiti di rappresentazione (mantissa limitata), può introdurre errori di accuratezza nei risultati delle operazioni

- Ad esempio, vediamo un esempio di calcolo erroneo, da cui possiamo desumere che la somma in virgola mobile non è sempre associativa:
  in generale, non è quindi vero che $x+(y+z) = (x+y)+z$

- Il fenomeno si può osservare quando dobbiamo sommare due numeri molto *grandi in valore assoluto*, ma di segno opposto, con altro un numero molto piccolo

  $x = 1.5_{10} \cdot 10^{38}$
  $y = -\ 1.5_{10} \cdot 10^{38}$ con x,y,z espressi in singola precisione
  $z = 1.0_{10}$

  $$\begin{aligned} x+(y+z) &= -1.5 \cdot 10^{38} + (1.5 \cdot 10^{38} + 1.0) \\ &= -1.5 \cdot 10^{38} + 1.5 \cdot 10^{38} = 0.0_{10} \end{aligned}$$

  $$\begin{aligned} (x+y)+z &= (-1.5 \cdot 10^{38} + 1.5 \cdot 10^{38}) + 1.0 \\ &= 0.0 + 1.0 = 1.0_{10} \end{aligned}$$

Codifica caratteri alfa-numerici

- I calcolatori, nonostante il nome italiano (in francese si chiamano ordinatori) sono spesso utilizzati per manipolare informazioni non numeriche.
- Si parla di caratteri "alfanumerici" per sottolineare che in un testo sono presenti:
  - caratteri alfabetici (a,b,c,d,…)
  - caratteri numerici (0,…,9)
  - segni di punteggiatura (!,?,…)
  - simboli particolari vario tipo (£, &, @, ...)
- I processori moderni non hanno istruzioni specifiche per testi. Quindi, si usano codifiche da testo a numeri interi
- Un testo è una sequenza di caratteri. I codici associano un numero intero ad ogni carattere.

Codifiche di caratteri

1968 ASCII.
Codice a 7 bit: 95 caratteri stampabili e 33 di controllo.

1980 Extended ASCII.
Varie estensioni a 8 bit, con simboli grafici e lettere accentate.

1991 Unicode.
Codice a 21 bit (> 1 milione di simboli). Attualmente (v. 13.0) definiti circa 150.000 caratteri!

https://www.unicode.org/charts/

Viene ulteriormente codificato in **UTF-8.**

1992 UTF-8.
Codifica di Unicode a lunghezza variabile (da 1 a 4 byte). Retro-compatibile con ASCII. UTF-8 è la codifica consigliata per XML e HTML.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68		D	100		d
5	5	[ENQUIRY]	37	25	%	69		E	101		e
6	6	[ACKNOWLEDGE]	38	26	&			F			f
7	7	[BELL]	39	27	'			G			g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51		3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52		4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE			5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]			6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

A: 100 0001

a: 110 0001

“0”: 011 0000

Codice ASCII

- Ai primi 32 numerali sono assegnati caratteri di controllo
  - null indica la fine di una stringa
  - carriage return porta il cursore su una nuova riga (andata a capo)
  - horizontal tab è l’usuale carattere tab
- Altro tipo:
  - Bell dovrebbe far suonare un cicalino

Caratteri Unicode

Code Charts — Tech Site | Site Map | Search

Unicode 13.0 Character Code Charts

SCRIPTS | SYMBOLS & PUNCTUATION | NAME INDEX

Find chart by hex code: [ ] Go   Help   Conventions   Terms of Use

Scripts

European Scripts

- *Armenian*
  - *Armenian Ligatures*
- *Carian*
- *Caucasian Albanian*
- *Cypriot Syllabary*
- *Cyrillic*
  - Cyrillic Supplement
  - Cyrillic Extended-A
  - Cyrillic Extended-B
  - Cyrillic Extended-C
- *Elbasan*
- *Georgian*
  - Georgian Extended
  - Georgian Supplement
- *Glagolitic*
  - Glagolitic Supplement
- *Gothic*
- *Greek*
  - Greek Extended

African Scripts

- *Adlam*
- *Bamum*
  - Bamum Supplement
- *Bassa Vah*
- *Coptic*
  - *Coptic in Greek block*
  - Coptic Epact Numbers
- *Egyptian Hieroglyphs (1MB)*
  - Egyptian Hieroglyph Format Controls
- *Ethiopic*
  - Ethiopic Supplement
  - Ethiopic Extended
  - Ethiopic Extended-A
- *Medefaidrin*
- *Mende Kikakui*
- *Meroitic*
  - Meroitic Cursive
  - Meroitic Hieroglyphs
- *N'Ko*

South Asian Scripts

- *Ahom*
- *Bengali and Assamese*
- *Bhaiksuki*
- *Brahmi*
- *Chakma*
- *Devanagari*
  - Devanagari Extended
- *Dives Akuru*
- *Dogra*
- *Grantha*
- *Gujarati*
- *Gunjala Gondi*
- *Gurmukhi*
- *Kaithi*
- *Kannada*
- *Kharoshthi*
- *Khojki*
- *Khudawadi*
- *Lepcha*

Indonesia & Oceania Scripts

- *Balinese*
- *Batak*
- *Buginese*
- *Buhid*
- *Hanunoo*
- *Javanese*
- *Makasar*
- *Rejang*
- *Sundanese*
  - Sundanese Supplement
- *Tagalog*
- *Tagbanwa*

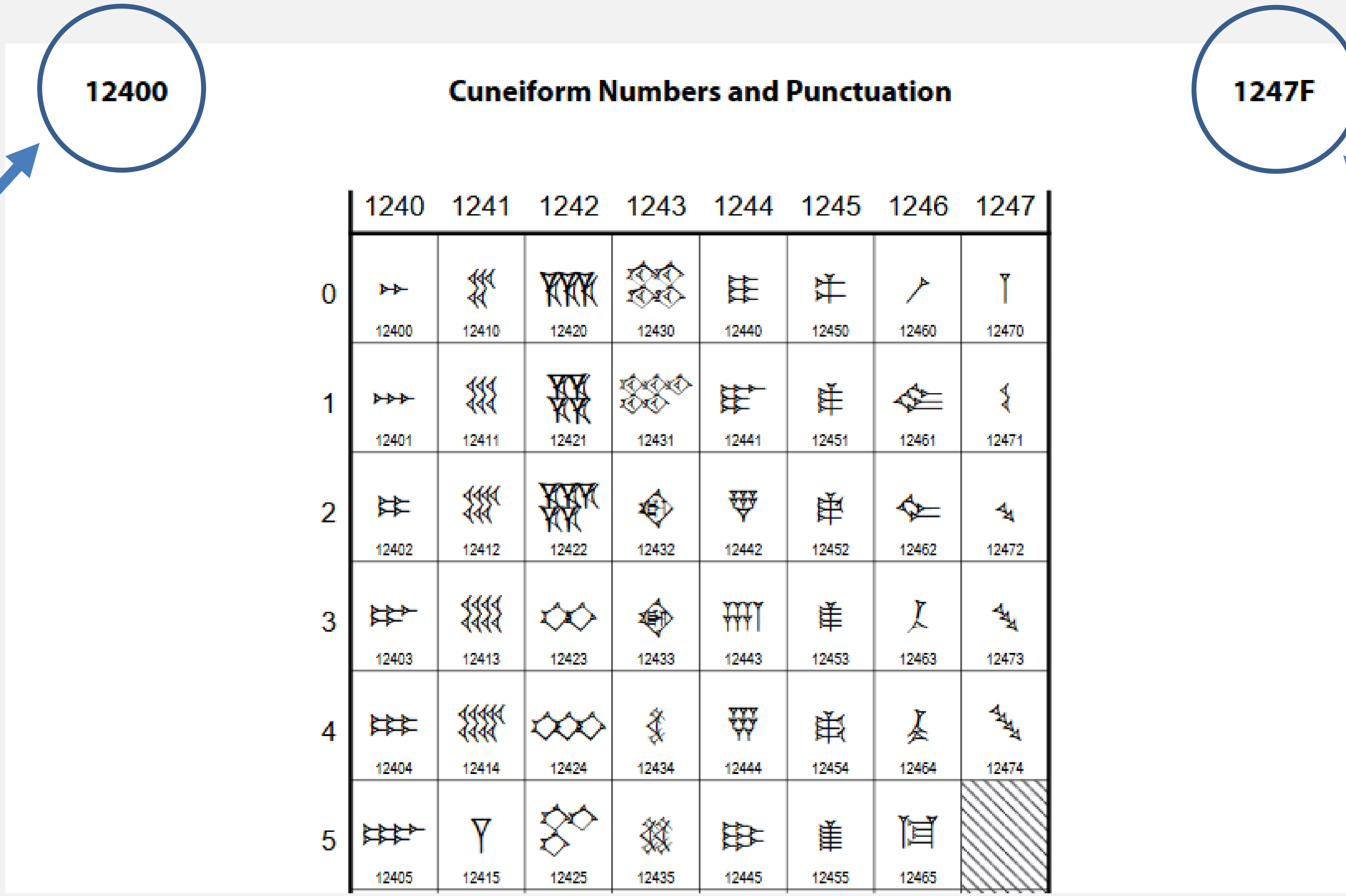
East Asian Scripts

- *Bopomofo*
  - Bopomofo Extended
- *CJK Unified Ideographs (Han)* (35MB)
  - CJK Extension A (6MB)
  - CJK Extension B (40MB)
  - CJK Extension C (3MB)

https://www.unicode.org/charts/

Caratteri Cuneiformi in Unicode

start



end

https://www.unicode.org/charts/PDF/U12400.pdf

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF[12]	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

La codifica UTF-8

- UTF-8 codifica ciascun carattere Unicode con una sequenza lunga da 1 a 4 byte. Il primo byte di un carattere indica quanto è lunga la sequenza:

Primo byte	Byte totali	Bit a disposizione del carattere
0xxx xxxx	1	7
110x xxxx	2	11
1110 xxxx	3	16
1111 0xxx	4	21

- Tutti i byte successivi nella sequenza hanno il formato 10xx xxxx

Esempio di UTF-8

- Considieriamo il simbolo dell’euro “€”, codice Unicode U+20AC

- E’ un codice di 16 bit, quindi richiede 3 byte in UTF-8
- Vediamo come i 16 bit vengono distribuiti su 3 byte da UTF-8:

	4 bit	6 bit	6 bit
0x20AC =	0010	000010	101100

- Codifica UTF-8:

1110 0010 **10**00 0010 **10**10 1100

3 byte

Il successo di UTF-8