La codifica binaria

A.S. 2026-2027

Classi Prime

Prof. Valentino Pietrobon



Attribuzione - Non commerciale - Non opere derivate
4.0 Internazionale
(CC BY-NC-ND 4.0)

Tu sei libero di:

Condividere — riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare questo materiale con qualsiasi mezzo e formato

Il licenziante non può revocare questi diritti fintanto che tu rispetti i termini della licenza.

Alle seguenti condizioni:

Attribuzione — Devi riconoscere una menzione di paternità adeguata, fornire un link alla licenza e indicare se sono state effettuate delle modifiche. Puoi fare ciò in qualsiasi maniera ragionevole possibile, ma non con modalità tali da suggerire che il licenziante avalli te o il tuo utilizzo del materiale.

NonCommerciale — Non puoi utilizzare il materiale per scopi commerciali.

Non opere derivate — Se remixi, trasformi il materiale o ti basi su di esso, non puoi distribuire il materiale così modificato.

Divieto di restrizioni aggiuntive — Non puoi applicare termini legali o misure tecnologiche che impongano ad altri soggetti dei vincoli giuridici su quanto la licenza consente loro di fare.

Note:

Non sei tenuto a rispettare i termini della licenza per quelle componenti del materiale che siano in pubblico dominio o nei casi in cui il tuo utilizzo sia consentito da una eccezione o limitazione prevista dalla legge.

Non sono fornite garanzie. La licenza può non conferirti tutte le autorizzazioni necessarie per l'utilizzo che ti prefiggi. Ad esempio, diritti di terzi come i diritti all'immagine, alla riservatezza e i diritti morali potrebbero restringere gli usi che ti prefiggi sul materiale.

Sommario

1 Dal bit al Byte 5
1.1 Il Bit 5
1.1.1 Il bit come cifra binaria 5
1.1.2 Il bit come quantità di informazione 5
1.2 Il Byte 6
1.2.1 I multipli del Byte 7
2 La codifica delle informazioni 7
2.1 La codifica binaria 8
3 La codifica dei numeri naturali 10
3.1 Sistemi numerici posizionali 10
3.2 Sistema numerico decimale 11
3.3 Sistema numerico binario senza segno 12
3.3.1 Conversione da binario senza segno a decimale 12
3.3.2 Conversione da decimale a binario senza segno 12
3.3.3 Valori massimi numeri binari senza segno 13
3.3.4 Operazioni aritmetiche 14
3.4 Sistema numerico in base 16 (esadecimale) 16
3.4.1 Da base 2 a base 16 16
3.4.2 Da base 16 a base 2 16
3.4.3 Da base 16 a base 10 17
3.4.4 Da base 10 a base 16 17
3.4.5 Valori massimi in base 16 17
4 La codifica dei numeri interi 18
4.1 Rappresentazione Modulo-Segno 18
4.1.1 Passaggio da 8 bit a 16 bit (Modulo-Segno) 19
4.1.2 Valori minimi e valori massimi (Modulo-Segno) 19
4.1.3 Overflow e Underflow (Modulo-Segno) 19
4.2 Rappresentazione in Complemento a 2 20
4.2.1 Passaggio da 8 bit a 16 bit (Complemento a 2) 21
4.2.2 Valore minimo e valore massimo (Complemento a 2) 21
4.2.3 Overflow e Underflow (Complemento a 2) 21
5 La codifica dei numeri razionali 23
5.1 Da base 10 a base 2 (numeri razionali) 23

5.2 Da base 2 a base 10 (numeri razionali) ... 24
5.2.1 Legame tra periodicità e base ... 24
5.3 Rappresentazione in virgola fissa ... 24
5.4 Rappresentazione in virgola mobile (IEEE 754) ... 25
5.4.1 Struttura di un numero in virgola mobile ... 25
5.4.2 Numeri a precisione singola (32 bit) ... 25
5.4.3 Esempio di Codifica ... 26
5.4.4 Formati più usati ... 27
5.4.5 Intervallo dell'esponente in base 10 ... 27
5.4.6 Esempio di decodifica ... 27
5.4.7 Approfondimento ... 28
6 Codifiche per il testo o come testo ... 31
6.1 La codifica ASCII ... 31
6.1.1 Esempio di codifica e decodifica ... 33
6.2 Codifica Base64 ... 33
6.3 I codici UCS-2 e UTF-8 ... 34
6.4 Allegato: ISO 8859-16 (Latin-10 South-Eastern European) ... 36
7 Sitografia ... 38

1 Dal bit al Byte

1.1 Il Bit

La parola bit, in Informatica e in Teoria dell'Informazione, ha due significati molto diversi, a seconda del contesto in cui rispettivamente la si usi.

- Un bit è **l'unità di misura dell'informazione** (dall'inglese "binary digit"), definita come la **quantità minima di informazione** che serve a discernere tra due possibili eventi equiprobabili.
- Un bit è **una cifra binaria**, ovvero uno dei due simboli del sistema numerico binario, classicamente chiamati **zero (0) e uno (1)**; in questo caso si può parlare di numero di 8, 16, 32... bit proprio come nella comune base dieci si parla di un numero di 2, 5, 6... cifre.

Esempio

La differenza di significato tra le due definizioni può riassumersi con una frase come: "la ricezione degli ultimi 18 bit (simboli binari) di messaggio ha aumentato la nostra informazione di 6 bit (quantità di informazione)". Considero la sequenza di bit 101101 e decido di trasmettere ogni bit triplicato cioè 111000111111000111. In questo caso ho ricevuto 18 bit ma questi portavano solo 6 bit di informazione.

1.1.1 Il bit come cifra binaria

In questo contesto il bit rappresenta l'unità elementare per definire lo stato logico di un circuito (ad esempio un circuito di memoria). Viene definito anche come unità elementare dell'informazione trattata da un elaboratore.

La rappresentazione del bit avviene attraverso i valori 0 e 1.

All'atto pratico, 2 valori sono troppo pochi per rappresentare più elementi di un insieme, per questo si raggruppano sequenze di bit in un'unica entità in modo che esse possano rappresentare valori in intervalli più ampi di quello consentito da un singolo bit.

Con 2 bit  si possono avere 4 valori rappresentabili: 00, 01, 10, 11

Questi raggruppamenti possono contenere un numero qualsiasi di bit anche se, generalmente, contengono un numero di cifre binarie pari a una potenza del 2 come ad esempio 2, 4, 8, 16, ecc.

Il più noto raggruppamento è il **Byte**, corrispondente a 8 bit, che costituisce l'unità di misura più utilizzata in campo informatico. Altri raggruppamenti di questo tipo sono i seguenti:

- **Nibble** 4 bit, la metà di un Byte
- **Word** di lunghezza variabile, corrisponde a 16 o 32 o 64 bit a seconda del contesto.

Il mondo Windows definisce il tipo **WORD di 16 bit**. Questa circostanza, unita alle normali esigenze di retrocompatibilità dei nuovi sistemi con i vecchi, fa sì che spesso la lunghezza di una Word venga fissata a 16 bit indipendentemente dall'hardware.

Per questo motivo si parla anche di:

- **double word** pari a 2 word (DWORD o LONGWORD) - 32 bit
- **quad word** pari a 4 word (QWORD) - 64 bit

1.1.2 Il bit come quantità di informazione

In questo contesto, un bit rappresenta l'unità di misura della quantità d'informazione.

Questo concetto di bit è stato introdotto dalla Teoria dell'Informazione di Claude Shannon nel 1948, ed è usato nella **compressione dati** e nelle **trasmissioni numeriche**.

Intuitivamente assegnare un valore a un bit equivale alla scelta tra due valori (sì/no, vero/falso, acceso/spento) quando questi hanno la stessa probabilità di essere scelti. In generale, per eventi non necessariamente equiprobabili, la quantità d'informazione di un evento rappresenta la "sorpresa" nel constatare il verificarsi di tale evento.

Per esempio, se un evento è certo, il suo verificarsi non sorprende nessuno, quindi il suo contenuto informativo è nullo, se invece un evento è raro, il suo verificarsi è sorprendente, quindi il suo contenuto informativo è alto.

Ma avremo modo di riprendere questi concetti in quinta quando tratteremo nel dettaglio la Teoria dell'informazione.

1.2 Il Byte

Un Byte, il cui nome deriva dalla parola inglese bite (boccone, morso), coniato anche per assonanza col termine "bit" ma rinominato sostituendo la 'i' con la 'y' per evitare confusioni di pronuncia, è una sequenza di bit.

Il termine Byte è stato coniato da Werner Buchholz[1] nel luglio 1956, all'inizio del progetto del computer IBM Stretch[2], ma è solo dal 1964 che il Byte è formato da 8 bit. Tali bit possono assumere $2^8 = 256$ possibili combinazioni associabili ai valori da 0 a 255.

Agli inizi degli anni 1960, esistevano due tipi di computer: quelli scientifici e quelli commerciali. Nei primi per la rappresentazione delle dieci cifre decimali, da '0' a '9', venivano utilizzati 4 bit (BCD – codice binario di cifra), mentre nei secondi venivano usati 6 bit ($2^6 = 64$ configurazioni) per rappresentare ogni carattere del set grafico stampabile. Il set di caratteri che comprendeva 26 caratteri alfabetici (solo maiuscole), 10 cifre numeriche, e svariati simboli speciali: + - * / (> ecc.

In seguito, per includere i caratteri di controllo e permettere ai dispositivi di comunicare fra loro, memorizzare, trasferire caratteri per la scrittura dei testi comprendendo pure i caratteri minuscoli, è stato introdotto il codice ASCII a 7 bit (2^7=128 configurazioni).

Successivamente, poiché 8 bit permettevano di contenere convenientemente due cifre decimali BCD di 4 bit ciascuna, è stato proposto e adottato, come codice di rappresentazione standard dall'IBM nei computer System/360, il codice EBCDIC, un codice da 8 bit, compatibile con l'attuale Byte.

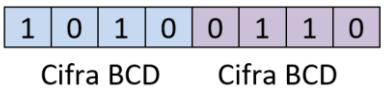


Figura 1 – Utilizzo di un Byte per contenere due cifre BCD

Un Byte ha una lunghezza utile per molti scopi:

- servono, ad esempio, circa 200 < 256 = 2^8 livelli di grigio per passare dal bianco al nero, perché l'occhio umano pensi di vedere una sfumatura continua,
- servono circa 200 < 256 = 2^8 caratteri diversi (lettere accentate comprese) per scrivere un qualsiasi testo di una lingua occidentale;
- servono circa 200 < 256 = 2^8 livelli di intensità per riprodurre un suono abbastanza fedele.

[1] https://it.wikipedia.org/wiki/Werner_Buchholz
[2] https://it.wikipedia.org/wiki/IBM_7030_Stretch

Il simbolo utilizzato per il Byte come unità di misura della quantità di informazione è la lettera **B**; mentre la lettera **b**, generalmente, è usata per indicare il bit (il cui simbolo standard sarebbe **bit** scritto per esteso).

Per comodità di calcolo i multipli del Byte vengono generalmente arrotondati a potenze del 2, invece che potenze del 10. Tale ambiguità ha portato l'IEC[3] a definire nuovi prefissi per multipli binari.

Questa ambiguità viene sfruttata a fini commerciali dai produttori di hard disk, le potenze del 10, ma poco in uso, in modo tale da far figurare capienze maggiori: un disco rigido di 80 GB di capacità equivale a circa 74,5 GiB.

1.2.1 I multipli del Byte

Nella tabella seguente sono elencati i prefissi ufficiali IEC, i prefissi decimali normalmente usati e la differenza percentuale tra i due.

Tabella 1 – Multipli del Byte

Nome IEC	Sigla	Quantità	Nome in uso	Sigla	Quantità formale	Differenza
byte	B	2^0=1 B	Byte	B	10^0=1 B	0
kibibyte	KiB	2^{10}=1024 B	Kilobyte	KB	10^3=1000 B	2,4%
mebibyte	MiB	2^{20}=1048576 B	Megabyte	MB	10^6=1000000 B	4,9%
gibibyte	GiB	2^{30}=1073741824 B	Gigabyte	GB	10^9 B	7,4%
tebibyte	TiB	2^{40}=1099511627776 B	Terabyte	TB	10^{12} B	10,0%
pebibyte	PiB	2^{50}=1125899906842624 B	Petabyte	PB	10^{15} B	12,6%
exbibyte	EiB	2^{60}=1152921504606846976 B	Exabyte	EB	10^{18} B	15,3%
zebibyte	ZiB	2^{70}=1180591620717411303424 B	Zettabyte	ZB	10^{21} B	18,1%
yobibyte	YiB	2^{80}=1208925819614629174706176 B	Yottabyte	YB	10^{24} B	20,9%

2 La codifica delle informazioni

I calcolatori elettronici utilizzano dati codificati in un unico formato adatto alla memorizzazione e alla manipolazione da parte di un esecutore automatico. Per trasformare qualsiasi tipo di dato (numeri, testo, immagini, suoni, istruzioni), si utilizzano, pertanto, successioni di simboli 0 e 1.

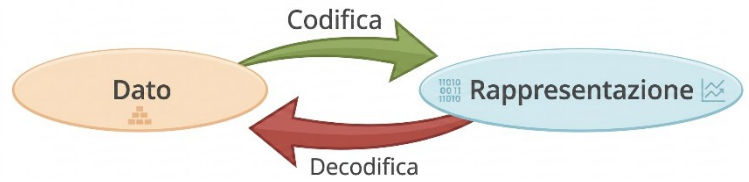


Figura 2 – Codifica e decodifica delle informazioni

Oltre ai simboli 0 e 1, è necessario avere un insieme di regole di composizione che associano sequenze di 0 e 1 ai singoli dati da rappresentare. La **relazione** che lega ogni singolo dato (**informazione senza contesto**) ad una sequenza di 0 e 1 (**rappresentazione**) è detta **codice** e l'operazione di trasformazione **codifica**.

Gli attuali calcolatori utilizzano circuiti detti bistabili, in grado di assumere uno stato scelto tra due differenti possibili (associati alla presenza o assenza di un determinato valore di tensione elettrica). Un alfabeto di questo tipo è chiamato "**alfabeto binario**" e si può considerare alla base dell'elaborazione dell'informazione. Per comodità si preferisce parlare di 0 e 1 invece che di assenza o presenza di tensione.

[3] Commissione Elettrotecnica Internazionale

2.1 La codifica binaria

La codifica binaria è utilizzata, quindi, per rappresentare qualsiasi informazione (caratteri alfanumerici, numeri, immagini, suoni, video, etc.) utilizzando un alfabeto limitato a soli due caratteri: 0 e 1, in quanto i circuiti di un elaboratore elaborano o memorizzano solo cifre binarie (corrispondenti ai due possibili stati di un circuito elettrico: aperto o chiuso).

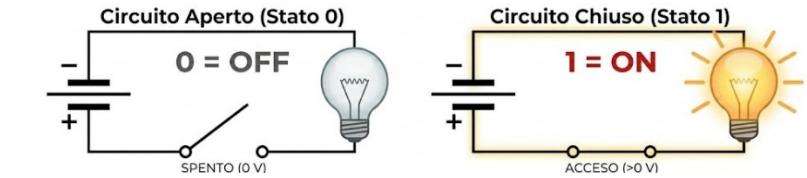


Figura 3 – Circuito elettrico a sinistra aperto (0 – lampadina spenta) e a destra chiuso (1 – lampadina accesa). Ogni circuito ha un alimentatore a sinistra, un interruttore in basso e una lampadina a destra.

Ogni elemento che assume un valore binario (un valore scelto tra due valori possibili come 0 e 1) viene indicato con il termine **bit** (da **bi**nary digi**t**, cifra binaria).

Un bit permette di rappresentare una scelta tra "si" e "no", due risposte che possono essere associate ai valori "0" e "1". Il passaggio della corrente elettrica con l'interruttore chiuso e la lampadina accesa rappresenta il simbolo 1, mentre l'assenza di corrente elettrica con l'interruttore aperto e la lampadina spenta rappresenta il simbolo 0.

Ovviamente, la rappresentazione di informazioni più complesse richiede l'uso di un numero maggiore di bit.

La risposta "si" o "no" a una domanda porta 1 bit di informazione. La risposta a due domande (che prevede una soluzione con 4 alternative: "sì-sì", "no-no", "sì-no", "no-sì") porta 2 bit di informazione e così via.

È necessario utilizzare più di 1 bit anche quando le informazioni che si vogliono rappresentare presentano più di due alternative. Ad esempio, i punti cardinali Nord, Sud, Est e Ovest richiedono anch'essi 2 bit per poter rappresentare le 4 possibili alternative. Questo caso è del tutto simile a quello di 2 risposte sì-no. Ad esempio, Est sarà memorizzato dai due bit aventi valori 1 e 0. Evidentemente l'ordine è importante in quanto i valori 0 e 1 identificano il Sud e non l'Est.

Tabella 2 – Esempio codifica binaria a 2 bit

Punti cardinali	Analogia due risposte	Codifica binaria
Nord	no-no	0 0
Sud	no-sì	0 1
Est	sì-no	1 0
Ovest	sì-sì	1 1

Supponiamo ora di dover memorizzare un giorno della settimana, ad esempio martedì.

L'insieme di tutti i valori assunti dall'informazione che si vuole rappresentare contiene 7 elementi diversi: lunedì, martedì, mercoledì, giovedì, venerdì, sabato e domenica.

Quindi non sono sufficienti 2 bit, in quanto abbiamo visto che con 2 bit abbiamo al massimo 4 sequenze di codifica, ma 3 bit saranno sufficienti? Con 3 bit è possibile avere $8 = 2^3$ sequenze di codifica e pertanto saranno sufficienti (una sequenza rimarrà inutilizzata).

Tabella 3 – Esempio codifica binaria a 3 bit

Giorni settimana	Codifica binaria
Lunedì	0 0 0
Martedì	0 0 1

Mercoledì	0 1 0
Giovedì	0 1 1
Venerdì	1 0 0
Sabato	1 0 1
Domenica	1 1 0
Non usato	1 1 1

Nota: la scelta della sequenza di bit da associare ad ogni giorno della settimana è del tutto arbitraria come pure la codifica non usata.

È possibile avere una regola per ottenere il numero di possibili sequenze di codifica dato il numero di bit? Sì, basta utilizzare le potenze del 2.

$2^{N°Bit}$ = N° max Codifiche

Utilizzando questa uguaglianza è possibile ricavare il numero minimo di bit necessari per codificare un insieme di elementi quando se ne conosce la cardinalità (il numero di elementi).

Supponiamo, ad esempio di dover codificare le lettere maiuscole del nostro alfabeto:

Lettere = {A, B, C, D, E, F, G, H, I , J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z}

L'insieme ha cardinalità 26. Quanti bit sono necessari per avere 26 codifiche?

Costruiamo la seguente tabella:

Tabella 4 – Corrispondenza numero di bit e numero massimo di codifiche

N° bit	N° massimo di codifiche
1	$2^1 = 2$
2	$2^2 = 4$
3	$2^3 = 8$
4	$2^4 = 16$
5	$2^5 = 32$

Come si vede chiaramente dalla tabella, 4 bit sono insufficienti per avere 26 codifiche, mentre con 5 bit si hanno 32 codifiche più che sufficienti per codificare i 26 caratteri.

A questo punto procediamo con la costruzione della "**Tabella di codifica**".

Tabella 5 – Esempio codifica binaria a 5 bit

Lettera	Codifica	Numero	Lettera	Codifica	Numero
A	00001	1	N	01110	14
B	00010	2	O	01111	15
C	00011	3	P	10000	16
D	00100	4	Q	10001	17
E	00101	5	R	10010	18
F	00110	6	S	10011	19
G	00111	7	T	10100	20
H	01000	8	U	10101	21
I	01001	9	V	10110	22
J	01010	10	W	10111	23
K	01011	11	X	11000	24
L	01100	12	Y	11001	25
M	01101	13	Z	11010	26

Come si può notare, non è necessario iniziare sempre dalla codifica fatta tutta di cifre 0. In realtà, non è nemmeno necessario utilizzare questa sequenza di codifiche specifica, ma poiché essa è esattamente la sequenza di codifica che utilizzeremo anche per codificare i numeri interi positivi, si preferisce utilizzare sempre questa sottintendendo che ad ogni lettera viene implicitamente associato un numero. Questa osservazione è alla base dell'affermazione che qualsiasi cosa è rappresentabile attraverso numeri.

3 La codifica dei numeri naturali

I numeri naturali sono caratterizzati dall'assenza del segno e pertanto tutti i bit disponibili possono essere utilizzati per rappresentare il numero.

La rappresentazione scelta dovrà rispettare le normali regole del calcolo studiate alla scuola primaria, quando imparavamo i metodi di calcolo per le 4 operazioni, solo che ora non si utilizzano 10 cifre ma solo 2, 0 e 1.

Inevitabilmente, questa limitazione avrà delle conseguenze, la più evidente sarà l'elevato numero di cifre necessarie per rappresentare i numeri decimali. Ad esempio, il numero decimale 10 sarà rappresentato dal numero binario 1010.

Per distinguere il numero decimale 10 dal numero binario 1010, si scriverà 10_{10} e 1010_2. In questo modo sarà chiaro che il primo è un numero decimale e il secondo un numero binario. Convenzionalmente, con i valori in base 10 si omette di indicare la base anche se questo potrebbe essere scambiato per il valore e non per la rappresentazione, ma visto che per la base 10 il valore e la rappresentazione coincidono, perché si utilizza la rappresentazione in base 10 anche per indicare il valore di un numero, allora l'ambiguità è irrilevante. Talvolta si preferisce scrivere il numero a parola per intenderne il valore e non la sua rappresentazione e quindi il valore di 10 è "dieci".

Se utilizzassimo i numeri romani, il valore dieci sarebbe espresso dalla codifica X, il valore cinquantasei dalla codifica LVI. Il sistema romano era un sistema non posizionale e questo rendeva i calcoli assai scomodi, mentre la bellezza del sistema posizionale è che i calcoli sono estremamente semplici e il metodo che abbiamo imparato, in realtà, non funziona solo con il sistema in base 10, ma con qualsiasi sistema posizionale a prescindere dalla base.

3.1 Sistemi numerici posizionali

Ma come si arriva alla codifica binaria 1010_2 data la codifica decimale 10_{10}?

Nel sistema di numerazione arabo (così chiamato per ragioni storiche, anche se la sua origine in realtà è indiana), la prima cifra da destra esprime il numero delle unità, la seconda quello delle decine, la terza quello delle centinaia, la quarta quello delle migliaia, e così via. Per esempio, il numero 555_{10} si legge: 5 centinaia, 5 decine, 5 unità (cinquecentocinquantacinque).

La stessa cifra 5 ha valore diverso a seconda della posizione in cui si trova, da cui il nome di sistema posizionale.



Figura 4 – Sistema posizionale, esempio valore "cinquecento cinquanta cinque"

In un sistema di numerazione non posizionale, invece, per esprimere in numero 555 si usano tre simboli diversi: ad esempio, con i numeri romani si scrive DLV (D=500, L=50, V=5).



Figura 5 – Sistema non posizionale, esempio valore "cinquecento cinquanta cinque"

I sistemi di numerazione posizionali hanno bisogno della cifra 0 per segnare i posti "vuoti". Ad esempio il numero 'cinquecentocinque' (5 centinaia, 0 decine, 5 unità) va scritto 505_{10}, con uno zero nella posizione delle decine; se non si mettesse lo zero, si otterrebbe 55_{10}, ottenendo un numero diverso.

Nei sistemi non posizionali non è necessario lo 0 perché l'uso di simboli diversi per unità, decine, centinaia lo rende inutile: il numero 'cinquecentocinque' si scrive DV e non si può confondere con 'cinquantacinque' che si scrive LV.

L'utilizzo della posizione per codificare il numero permette di usare un numero minore di simboli: infatti il sistema di numerazione indiano è in grado di rappresentare numeri grandi a piacere utilizzando solamente 10 simboli.

L'adozione della notazione posizionale ha permesso di definire algoritmi relativamente semplici e indipendenti dalla dimensione dei numeri per effettuare le operazioni elementari come l'addizione, la moltiplicazione o l'elevamento a potenza.

Un sistema numerico posizionale è definito da una successione di potenze. In formula:

$$c_4c_3c_2c_1 = c_4 \cdot b^3 + c_3 \cdot b^2 + c_2 \cdot b^1 + c_1 \cdot b^0$$

Il numero b si dice **base del sistema di numerazione**.

Il sistema arabo ha base 10 (perciò viene detto sistema numerico decimale), ma questo non vuol dire che si possa utilizzare una base diversa da 10 per rappresentare un numero, ad esempio base 4-

Ad esempio, la rappresentazione 2103 in base 4 rappresenta il valore:

$$2103_4 = 2x4^3+1x4^2+0x4^1+3x4^0 = 2x64 + 1x16 + 0x4 + 3x1 = 128 + 16 + 3 = 147$$

Con l'avvento dei calcolatori digitali, si è passati ai seguenti sistemi di numerazione posizionale

- b=2: sistema numerico binario
- b=8: sistema numerico ottale
- b=16: sistema numerico esadecimale

3.2 Sistema numerico decimale

Prima dei Greci, già i Sumeri e le popolazioni indoeuropee facevano uso di una numerazione in base 10, molto prima che fossero effettivamente inventate le cifre, e certamente molto prima che si diffondesse la scrittura posizionale.

Il successo di questa particolare base di numerazione si deve a varie cause:

- la base 10 permette di scrivere quantità anche considerevoli con poche cifre;
- i simboli da ricordare sono pochi, solo 10;
- il 10 è un numero intrinsecamente legato al corpo dell'uomo, 10 dita delle mani utilizzabili per fare i calcoli. Come notò Aristotele, l'uso del sistema decimale non fu altro che il risultato del fatto che l'uomo è nato con 10 dita dei piedi e 10 dita delle mani.

Vediamo come calcolare il valore del numero 245_{10}. Trattandosi di un numero in base 10, utilizzeremo le potenze del 10 per pesare opportunamente tutte e 3 le cifre che compongono il numero:

$245_{10} = 2 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0 = 2 \cdot 100 + 4 \cdot 10 + 5 \cdot 1 = 245$

Nota: Il valore 245, scritto a destra, è il valore, mentre 245_{10} è la sua rappresentazione in base 10, ma poiché noi utilizziamo i numeri in base 10 per indicare i valori, sembra di avere a che fare con la stessa cosa, anche se uno è la rappresentazione in base 10 e l'altro è il valore numerico vero e proprio.

3.3 Sistema numerico binario senza segno

Un numero binario senza segno è una sequenza di cifre binarie (dette bit). Ogni cifra in posizione n si considera moltiplicata per 2^n, anziché per 10^n, come avviene nella numerazione decimale.

Nella seguente tabella vengono confrontate le rappresentazioni binarie, esadecimale e decimale dei numeri compresi tra 0 e 15:

Tabella 6 – Codifica binaria e esadecimale per i numeri da 0 a 15

Binario	Esadecimale	Decimale	Binario	Esadecimale	Decimale
0	0	0	1000	8	8
1	1	1	1001	9	9
10	2	2	1010	A	10
11	3	3	1011	B	11
100	4	4	1100	C	12
101	5	5	1101	D	13
110	6	6	1110	E	14
111	7	7	1111	F	15

3.3.1 Conversione da binario senza segno a decimale

La formula per convertire un numero da binario a decimale (dove d_n indica la cifra (digit) di posizione n all'interno del numero, contata da destra verso sinistra iniziando da 0) è:

$$d_n \cdot 2^n + d_{n-1} \cdot 2^{n-1} + \ldots + d_1 \cdot 2^1 + d_0 \cdot 2^0 = N_{10}$$

Ad esempio:

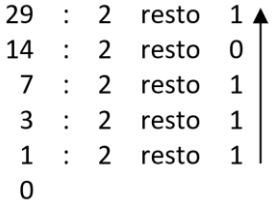
$1001_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 8 + 1 = 9_{10}$.

L'utilizzo dei numeri binari non è ristretto esclusivamente alla rappresentazione dei numeri interi positivi. Adottando alcune convenzioni, è possibile rappresentare numeri interi e numeri razionali.

3.3.2 Conversione da decimale a binario senza segno

La formula per convertire un numero da decimale a binario utilizza la divisione per 2 ripetutamente.

Supponiamo di dover rappresentare il numero 29 utilizzando la codifica binaria, allora avremo:



Il risultato di ogni divisione sarà riportato sotto al numero diviso, ad esempio 29 : 2 = 14 resto 1, e così via.

La codifica binaria sarà ottenuta leggendo in ordine, dal basso verso l'alto, tutti i resti delle divisioni. In questo caso $29 = 11101_2$.

Omettiamo la colonna di tutti 2 e tenendo solo la colonna dei risultati delle divisioni e dei resti:

72	0
36	0
18	0
9	1
4	0
2	0
1	1
0	

Quindi avremo: $72 = 1001000_2$.

Come abbiamo visto in precedenza, ogni cifra binaria ha un peso che corrisponde ad una potenza del due, in particolare avremo:

Tabella 7 – Valore associato ad ogni bit della codifica binaria per i primi 8 bit

valore	Potenza del 2	Cifra binaria corrispondente
1	2^0	1
2	2^1	10
4	2^2	100
8	2^3	1000
16	2^4	10000
32	2^5	100000
64	2^6	1000000
128	2^7	10000000

Pertanto, sarebbe possibile trattare i numeri direttamente e ricavare la codifica binaria.

Facciamo un esempio:

$54 = 32 + 22 = 32 + 16 + 6 = 32 + 16 + 4 + 2 = 110110_2$.

Ovviamente questo approccio richiede un minimo di familiarità con le potenze del due, ma ci evita la sequenza di divisioni per due sostituendola con una serie di somme ripetute.

3.3.3 Valori massimi numeri binari senza segno

Quando si prevede di memorizzare un numero espresso in cifre binarie, esso andrà a occupare un numero di bit ben definito, di solito 8, 16, 32, 64, 80, ecc.

Questo significa, se consideriamo il caso di 8 bit, che solo 8 cifre binarie saranno disponibili.

Se il numero minimo rappresentabile con 8 bit è facile: **$0 = 00000000_2$**, quale sarà il numero massimo rappresentabile con $11111111_2 = ?$

Visto che ogni cifra ha il suo peso all'interno del numero, basterà sommare i valori delle otto potenze del 2 (vedi tabella) ottenendo come risultato **128+64+32+16+8+4+2+1=255**.

Quindi **$255 = 11111111_2$**.

È possibile ricavare questo numero in modo diretto senza dover effettuare la somma di tutti i valori della tabella? Sì, basta notare che $11111111_2 + 1_2 = 100000000_2 = 256 = 2^8$ e, pertanto, **$11111111_2 = 100000000_2 - 1 = 256 - 1 = 255$**.

E se i bit fossero 10? Come valore minimo avremmo sempre **0** e come valore massimo **$2^{10} - 1 = 1024 - 1 = 1023$**.

3.3.4 Operazioni aritmetiche

I numeri binari possono essere messi in relazione tramite operazioni aritmetiche, con regole simili a quelle del sistema decimale. Le quattro operazioni aritmetiche eseguibili sono: somma, sottrazione, moltiplicazione e divisione. Per rendere possibili le operazioni aritmetiche e l'elaborazione dei segnali digitali è necessario esprimere i numeri binari tenendo conto del loro segno.

La somma

La somma fra due o più numeri binari è analoga a quella riferita ai numeri decimali. La regola applicata in questo caso è la seguente:

- 0+0=0
- 0+1=1
- 1+0=1
- 1+1=0 con riporto di 1 alla colonna verso sinistra

Quando si ha un riporto si aggiunge 1 sulla colonna di sinistra (quella più significativa) e si procede rispettando la regola della somma.

Riporto	1	1						
		1	1	0	1	+	13	+
			1	0	0	=	4	=
	1	0	0	0	1		17	

Riporto	1	1	1	1				
		1	1	0	1	+	13	+
			1	1	1	=	7	=
	1	0	1	0	0		20	

La sottrazione

La sottrazione nel sistema binario si svolge nel modo seguente:

- 0-0=0
- 0-1=1 con prestito -1 dalla colonna a sinistra
- 1-0=1
- 1-1=0

Quando si ha un prestito si sottrae 1 dalla colonna di sinistra (quella più significativa) e si procede rispettando la regola della differenza. Se sulla colonna di sinistra non si può concedere il prestito perché la cifra è 0, esso si trascina alla colonna successiva verso sinistra finché non si restituisce il prestito.

Prestito	-1	-1						
	1	1	0	1	-	13	-	
		1	1	0	=	6	=	
		1	1	1		7		

Prestito		-1	-1					
	1	1	0	0	0	-	24	-
	1	0	0	1	0	=	18	=
	0	0	1	1	0		6	

La moltiplicazione

La regola del prodotto binario segue esattamente quella della moltiplicazione di due numeri decimali. Infatti, si ha:

- $0 \times 0 = 0$
- $0 \times 1 = 0$
- $1 \times 0 = 0$
- $1 \times 1 = 1$

Quindi si tratta di procedere con i prodotti e poi con la somma dei vari prodotti parziali.

			1	1	1	0	x		1	4	x
				1	0	1	=			5	=
			1	1	1	0	+		7	0	
		0	0	0	0	-	+				
	1	1	1	0	-	-	=				
1	0	0	0	1	1	0					

Si noti che i prodotti parziali sono semplicemente o il primo numero o una sequenza di zeri.

Vediamo un esempio più complesso e riserviamo pure una riga per i riporti delle somme:

						1	1	0	1	1	x		2	7	x
						1	0	1	1	0	=		2	2	=
rip	1	1	1	2	1	1	1					5	9	4	
						0	0	0	0	0	+				
					1	1	0	1	1	-	+				
				1	1	0	1	1	-	-	+				
			0	0	0	0	0	-	-	-					
		1	1	0	1	1	-	-	-	-	=				
	1	0	0	1	0	1	0	0	1	0					

Divisione

L'operazione della divisione rispetta la regola:

$$D = Q \times d + R$$

dove D è il dividendo, d il divisore, Q è il quoziente ed R il resto.

Per eseguire una divisione si può procedere tramite il metodo tradizionale in cui si procede per sottrazioni successive tra il dividendo e il divisore d fino a che il risultato diventi inferiore al divisore. Il risultato finale rappresenta il resto mentre il numero di sottrazioni rappresenta il quoziente.

Calcoliamo 22 : 4 = 5 resto 2

1	0	1	1	0	:	1	0	0	=	1	0	1
1	0	0										
		1	1	0								
		1	0	0								
			1	0								
			resto									

Nel calcolo ci siamo fermati alla parte intera ma nulla vieta di continuare la divisione con le stesse modalità utilizzate per i numeri in base dieci e determinare la parte decimale.

Vediamo ora un altro esempio 363 : 9 = 40 resto 3

```
1 0 1 1 0 1 0 1 1 : 1 0 0 1 = 1 0 1 0 0 0
1 0 0 1
      1 0 0 1
      1 0 0 1
            0 0 1 1
```

3.4 Sistema numerico in base 16 (esadecimale)

Il sistema numerico esadecimale (spesso abbreviato come esa o hex) è un sistema posizionale in base 16, che utilizza 16 simboli invece di 10. Per il sistema esadecimale si usano i simboli da 0 a 9 per le prime dieci cifre, e poi le lettere da A, B, C, D, E, F per le successive cifre, per un totale di 16 simboli. Di seguito la tabella con le rappresentazioni esadecimali, decimali e binarie dei numeri da 0 a 15.

Tabella 8 – Codifiche binarie corrispondenti alle 16 cifre della base 16

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

Perciò il numero decimale 143, la cui rappresentazione binaria è 1000 1111_2, può essere scritto come 8F in esadecimale.

Il sistema esadecimale è molto usato in informatica, per la sua relazione diretta tra una cifra esadecimale e quattro cifre binarie. È spesso usato come intermediario, oppure come sistema numerico a sé stante.

Per esempio, è possibile esprimere un byte con esattamente due cifre esadecimali (invece che con 3 decimali, e lasciando gran parte dell'intervallo non utilizzato). È interessante, infatti, notare come ogni cifra esadecimale corrisponda a un nibble, cioè a un numero binario di quattro cifre.

3.4.1 Da base 2 a base 16

Consideriamo il numero 113, rappresentato in binario esso risulta essere 1110001_2. A questo punto, per ricavare la rappresentazione esadecimale, bisognerà suddividere le cifre binarie in gruppi di 4 partendo da destra in modo analogo a quanto fatto per passare alla codifica ottale.

$1110001_2 = 111|0001_2 = 0111|0001_2 = 71_{16}$

3.4.2 Da base 16 a base 2

Il processo inverso è facilmente intuibile, se ad ogni cifra esadecimale corrispondono 4 cifre binarie, basterà sostituire tutte le cifre esadecimali con gli equivalenti gruppi di 4 bit della base 2.

$71_{16} = 0111|0001_2 = 1110001_2$

3.4.3 Da base 16 a base 10

Un metodo per convertire un numero esadecimale in decimale è quello di moltiplicare le sue cifre per le potenze della base 16 corrispondenti alle posizioni sapendo che per la prima posizione abbiamo 16^0, per la seconda 16^1, e così via.

Ad esempio:

$4F_{16} = 4 \cdot 16^1 + F \cdot 16^0$

dove F vale 15 e $16^0 = 1$. Allora

$4F_{16} = 4 \cdot 16^1 + 15 \cdot 1 = 4 \cdot 16 + 15 \cdot 1 = 64 + 15 = 79.$

Un altro esempio:

$FB3_{16} = F \cdot 16^2 + B \cdot 16^1 + 3 \cdot 16^0 = 15 \cdot 256 + 11 \cdot 16 + 3 \cdot 1 = 3840 + 176 + 3 = 4019.$

3.4.4 Da base 10 a base 16

L'operazione inversa - da decimale ad esadecimale - si realizza con una serie di divisioni successive. Si utilizza la divisione con resto.

Vediamo un esempio:

79/16 = 4 resto 15

Possiamo procedere così come abbiamo fatto per base 2 anche se le divisioni sono più complesse come pure il calcolo del resto.

```
79 | 15
 4 | 4
 0 |
```

Infine, si deve comporre il numero in sistema esadecimale: il 4 è indicato dal simbolo 4, il 15 dal simbolo F, quindi $79 = 4F_{16}$.

Nota: Poiché le divisioni per 16 sono scomode, il più delle volte si preferisce trasformare il numero decimale in numero binario e poi il binario in numero esadecimale raggruppando le cifre binarie a quattro a quattro.

3.4.5 Valori massimi in base 16

Anche in questo caso, è importante chiarire quali siano i valori massimi ottenibili al variare del numero di cifre utilizzabili. Per quanto riguarda il valore minimo anche in questo caso vale 0.

La formula generale è la seguente:

Valore massimo = $16^n - 1$

con n il numero di cifre disponibili.

Tabella 9 – Valori massimi al variare delle cifre disponibili per una rappresentazione in base 16

Numero cifre	Valore massimo
1	$16^1 - 1 = 15$
2	$16^2 - 1 = 255$
3	$16^3 - 1 = 4095$
4	$16^4 - 1 = 65535$
…	…

4 La codifica dei numeri interi

Il metodo più comune di rappresentare i numeri consiste nell'utilizzare una stringa di bit, usando il sistema binario. La lunghezza o **"precisione"** è il numero di bit impiegato nella rappresentazione. Un numero con precisione n, come abbiamo visto per i numeri naturali, può rappresentare fino a 2^n valori: da 0 a 2^n-1.

Esistono tre modi differenti per rappresentare i numeri negativi in un sistema binario. Il più comune è il "complemento a due", che permette di rappresentare i numeri da $-2^{(n-1)}$ a $2^{(n-1)} - 1$. Questo metodo è conveniente perché crea una corrispondenza biunivoca tra valori e rappresentazioni, e perché nelle operazioni di addizione e sottrazione non è necessario distinguere tra valori con segno e senza. Gli altri metodi usati sono quelli del "segno e valore assoluto" e dello "scostamento".

4.1 Rappresentazione Modulo-Segno

La rappresentazione in segno e modulo, o rappresentazione con grandezza e segno, è una rappresentazione dei numeri relativi in base 2, che estende il sistema numerico binario dei numeri naturali per rappresentare anche i numeri negativi.

Questo è il modo più semplice per rappresentare e distinguere numeri positivi e negativi: al numero binario vero e proprio viene anteposto un bit che, per convenzione, assume il valore 0 se il numero è positivo oppure il valore 1 se il numero è negativo.

Esempi:
- $+3_{10} = 0011_2$
- $-3_{10} = 1011_2$

Il primo grande difetto di questa rappresentazione è quello di avere due modi per scrivere il numero 0. Avremo infatti 0000 e 1000 con il significato rispettivamente pari a +0 e -0.

Tabella 10 – Codifiche Segno e Modulo con 4 bit

Valore	Codifica	Valore	Codifica
+0	0000	-0	1000
+1	0001	-1	1001
+2	0010	-2	1010
+3	0011	-3	1011
+4	0100	-4	1100
+5	0101	-5	1101
+6	0110	-6	1110
+7	0111	-7	1111

Si noti che il numero di codifiche disponibili è stato suddiviso in due parti uguali: delle 16 disponibili 8 sono andate ai numeri positivi e 8 ai numeri negativi.

Un altro svantaggio lo si ha nell'operazione di somma (e conseguentemente di sottrazione) tra due numeri, in cui, definiti i due numeri A e B, si procede secondo le seguenti regole:

- con A > 0 e B > 0 --> A + B
- con A > 0 e B < 0 --> A - |B|
- con A < 0 e B > 0 --> B - |A|
- con A < 0 e B < 0 --> - (|A| + |B|)

L'idea è quella di togliere il segno, svolgere il calcolo e reintrodurlo alla fine.

Osservazione: poiché i valori negativi si caratterizzano per un 1 come cifra più significativa (a sinistra), assume un'importanza basilare il numero di bit che si utilizzano per rappresentare il numero.

4.1.1 Passaggio da 8 bit a 16 bit (Modulo-Segno)

Si ricordi, ad esempio, che il numero 80 è codificato in binario così: 1010000_2.

Consideriamo ora il numero -80, esso differisce dal numero 80 per la presenza di un ulteriore 1 a sinistra degli altri bit. Sicuramente sarà necessario utilizzare almeno 8 bit per rappresentare tale numero, visto che 7 ne servono per rappresentare il modulo del numero, cioè il valore 80 senza segno.

1	1	0	1	0	0	0	0

Ma se il numero di bit utilizzati sarà diverso (di solito un multiplo di 8), ad esempio 16, la rappresentazione del numero sarà diversa.

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

Pertanto, risulta essenziale, prima di andare a definire la rappresentazione binaria di un numero intero, conoscere il numero di bit disponibili per la rappresentazione dello stesso.

4.1.2 Valori minimi e valori massimi (Modulo-Segno)

Il numero di bit disponibili darà anche informazioni sul minimo e massimo numero rappresentabile.

Infatti, se N è il numero totale di bit, N-1 saranno i bit utilizzati per rappresentare il numero senza segno e il valore massimo rappresentabile sarà 2^{N-1}-1. Il valore minimo sarà lo stesso del valore massimo preceduto dal segno meno $-(2^{N-1}-1)$.

Di seguito, riportiamo il valore minimo e il valore massimo rappresentabile con la codifica segno e modulo al variare del numero di bit utilizzabili:

Tabella 11 – Valore minimo e valore massimo rappresentabile al variare del numero di bit con la codifica segno e modulo

N: bit	Valore minimo	Valore massimo
2	-1	+1
3	-3	+3
4	-7	+7
5	-15	+15
6	-31	+31
7	-63	+63
8	-127	+127

4.1.3 Overflow e Underflow (Modulo-Segno)

Cosa accade se sommiamo 1 al massimo valore rappresentabile o se sottraiamo 1 al minimo numero rappresentabile? Ovviamente quello che otteniamo è un valore insensato.

0	1	1	1	1	1	1	1	+
0	0	0	0	0	0	0	1	=
0	0	0	0	0	0	0	0	

Quindi +127 + 1 = +0 ma si commette un errore di **Overflow**: si è superato il massimo numero rappresentabile.

1	1	1	1	1	1	1	1	+
1	0	0	0	0	0	0	1	=
1	0	0	0	0	0	0	0	

In questo caso -127 – 1 = -127 + (-1) = -0 ma si commette un errore di Underflow: si è superato il minimo numero rappresentabile.

Si capisce meglio la cosa se riportiamo in una tabella le codifiche estreme negative e positive.

0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1
0	1	1	1	1	1	1	0
0	1	1	1	1	1	0	1
0	1	1	1	1	1	0	0
…	…	…	…	…	…	…	…
1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0

4.2 Rappresentazione in Complemento a 2

Il complemento a 2 (in inglese two's complement) è il metodo più diffuso per rappresentare numeri con segno. La sua enorme diffusione è data dal fatto che i circuiti di addizione e sottrazione non devono esaminare il segno di un numero rappresentato per determinare quale delle due operazioni sia necessario applicare, permettendo tecnologie più semplici: si utilizza un solo circuito, il sommatore, sia per l'addizione che per la sottrazione.

Con il complemento a 2 tutti i numeri che cominciano per "1" sono numeri negativi, mentre tutti i numeri che cominciano con uno "0" sono numeri positivi.

Un numero binario positivo si può rendere negativo invertendo i bit e sommando 1 alla rappresentazione risultante dalla inversione. Proviamo ad applicare la regola. Fissiamo come dimensione della rappresentazione 8 bit e andiamo a determinare la rappresentazione in complemento a 2 del numero -1.

Tabella 12 – Passi per il calcolo del complemento a 2 di -1

Passi	Commento	Codifica
1	Valore 1	00000001
2	Complemento	11111110
3	+1	11111111

Vediamo se si ottiene la stessa rappresentazione per il valore -1 applicando le normali regole di calcolo per la sottrazione e impostiamo il calcolo 0 - 1 = -1

	2	2	2	2	2	2	2	2	Prestito
-1	-1	-1	-1	-1	-1	-1	-1		
	0	0	0	0	0	0	0	0	-
								1	=
	1	1	1	1	1	1	1	1	

Vediamo ora la regola inversa per passare da una rappresentazione binaria di un numero negativo, per quelli positivi la comprensione del valore è immediata.

Le operazioni da eseguire dovranno essere fatte esattamente nell'ordine inverso. Supponiamo di considerare il numero binario 10111001

Tabella 13 – Passi per il calcolo del valore di un numero in complemento a 2

Passi	Commento	Codifica

1	Valore negativo	10111001
2	-1	10111000
3	Complemento	01000111

ma 01000111_2 = 64+4+2+1 = 71, quindi 10111001_2 = -71

Una riflessione da fare, anche per questo tipo di rappresentazione, riguarda i valori minimo e massimo rappresentabili e gli eventuali errori che si commettono nei calcoli.

4.2.1 Passaggio da 8 bit a 16 bit (Complemento a 2)

Consideriamo nuovamente il numero -71, esso differisce dal numero 71 per la presenza di un ulteriore 1 a sinistra degli altri e per la complementazione delle cifre ad eccezione della prima. La rappresentazione sarà la seguente:

1	0	1	1	1	0	0	1

Ma se il numero di bit utilizzati sarà diverso (di solito un multiplo di 8), ad esempio 16, la rappresentazione del numero sarà diversa.

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

Come si può vedere, se con i numeri positivi gli 8 bit aggiuntivi devono essere riempiti con degli 0, nel caso di numeri negativi vanno riempiti con degli 1 che poi diventeranno 0 quando si andrà a complementare il numero per capire il valore rappresentato.

4.2.2 Valore minimo e valore massimo (Complemento a 2)

Se i valori positivi hanno come primo bit un bit con valore 0, allora il massimo numero rappresentabile sarà quello con uno 0 e poi tanti 1 quanti sono il numero di bit disponibili meno 1 (utilizzato per lo 0 iniziale). Nel caso di 8 bit a disposizione si ha:

Valore massimo = 011111111 = 127 = 2^{n-1}-1

Il valore minimo inizierà con un bit a 1 e tutti gli altri (se -1 = 11111111) a zero:

Valore minimo = 10000000 = -128 = -2^{n-1}

Di seguito, riportiamo il valore minimo e il valore massimo rappresentabile con la codifica in complemento a 2 al variare del numero di bit utilizzabili:

Tabella 14 – Valore minimo e valore massimo rappresentabile al variare del numero di bit con la codifica in complemento a 2

N: bit	Valore minimo	Valore massimo
2	-2	+1
3	-4	+3
4	-8	+7
5	-16	+15
6	-32	+31
7	-64	+63
8	-128	+127

4.2.3 Overflow e Underflow (Complemento a 2)

Ma che succede se con un calcolo si ottiene un risultato che supera il valore massimo o il valore minimo rappresentabile?

Non potendo aggiungere bit per compensare le esigenze del calcolo, quello che accade è che si sconfina nella zona dei numeri di segno opposto.

Per semplicità consideriamo 4 bit e i numeri che con essi si possono rappresentare e rappresentiamoli disposti lungo una circonferenza.

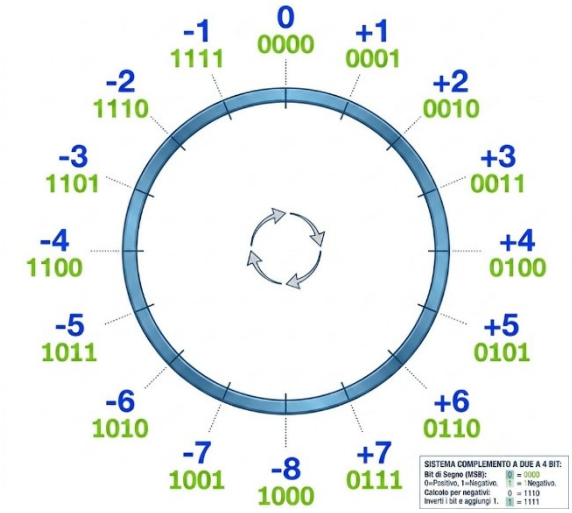


Figura 6 – Sistema Complemento a 2 con 4 bit

Come si può notare, dopo il 7 c'è il -8, per cui se a 7 sommo 1 il risultato non sarà 8 (valore corretto) ma -8 (valore errato) e in questo caso commetterò un errore di overflow (straripamento verso l'alto).

Se sottraggo -1 a -8 non ottengo -9 (valore corretto) ma 7 (valore errato) e in questo caso commetterò un errore di underflow (straripamento verso il basso).

5 La codifica dei numeri razionali

Un numero razionale si presenta, normalmente, con una parte intera e una parte frazionaria che il più delle volte avrà una prima parte detta antiperiodo e una seconda parte detta periodo. La presenza di quest'ultima parte implica che il numero debba essere rappresentato da infinite cifre. Molto spesso, erroneamente, si pensa che la presenza o meno del periodo dipenda dal valore del numero e non dalla base utilizzata per rappresentare il numero stesso.



Come vedremo, invece, moltissimi numeri razionali che in base dieci non hanno il periodo, in base due avranno un periodo anche abbastanza lungo.

5.1 Da base 10 a base 2 (numeri razionali)

Ma vediamo, innanzi tutto come esprimere un numero razionale in base 2.

- **La parte intera** verrà convertita come si faceva con i numeri interi con codifica segno e modulo: procedendo con le divisioni per due.
- **La parte frazionaria** sarà trattata in modo diverso.

Poiché le cifre binarie a destra della virgola rappresentano ancora potenze del 2 ma con esponente negativo:

$$0{,}1111_2 = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = 1/2 + 1/4 + 1/8 + 1/16 =$$
$$= 0{,}5 + 0{,}25 + 0{,}125 + 0{,}0625 = 0{,}9375$$

procederemo moltiplicando per 2 la parte frazionaria e ogni volta andremo ad aggiungere alla rappresentazione binaria la cifra intera ottenuta dalla moltiplicazione che sarà o 0 o 1 e ripeteremo la moltiplicazione per 2 solo sulla parte frazionaria rimanente.

Consideriamo, ad esempio il valore 7,625.

Parte intera:

$7 = 111_2$

Parte decimale:

0,625 x 2 = 1,25
0,25 x 2 = 0,5
0,5 x 2 = 1 Fine
$0{,}625 = 0{,}101_2$

$7{,}625 = 111{,}101_2$

Vediamo un altro esempio. Consideriamo in numero 13,1.

Parte intera:

$13 = 1101_2$

Parte decimale:

0,1 x 2 = 0,2
0,2 x 2 = 0,4
0,4 x 2 = 0,8
0,8 x 2 = 1,6
0,6 x 2 = 1,2 si torna al caso 0,2

$13{,}1 = 1101{,}\overline{00011}_2$

5.2 Da base 2 a base 10 (numeri razionali)

Vediamo alcuni esempi visto che il metodo è già stato esposto precedentemente.

Esempi

- $1101,11_2 = 2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-2} = 8 + 4 + 1 + 0,5 + 0,25 = 13,75$
- $10110,011_2 = 2^4 + 2^2 + 2^1 + 2^{-2} + 2^{-3} = 16 + 4 + 2 + 0,25 + 0,125 = 22,375$
- $11001,1011_2 = 2^4 + 2^3 + 2^0 + 2^{-1} + 2^{-3} + 2^{-4} = 16 + 8 + 1 + 0,5 + 0,125 + 0,0625 = 25,6875$

5.2.1 Legame tra periodicità e base

Come avevamo già accennato la periodicità dipende dalla base scelta e non dal valore rappresentato.

Ad esempio, il numero $0,\overline{1}$ che può essere scritto utilizzando la frazione generatrice $1/9 = 9^{-1}$, risulta essere rappresentato da 0,1 in base 9.

Infatti, 0,1 in base 9 rappresenta il valore $9^{-1} = 1/9 = 0,\overline{1}$.

5.3 Rappresentazione in virgola fissa

A questo punto chiariamo meglio i possibili tipi di rappresentazione per i numeri razionali:

- Rappresentazione in virgola fissa
- Rappresentazione in virgola mobile (vedremo l'anno prossimo)

Nella rappresentazione in virgola fissa si decide di assegnare un certo numero di bit alla parte intera e un certo numero di bit a quella frazionaria.

Ad esempio, supponiamo di utilizzare 8 bit e di assegnare 4 bit alla parte intera e 4 bit alla parte decimale. Sappiamo già che non potremo rappresentare numeri negativi e che il valore minimo sarà 0 e il valore massimo sarà 15,9375 corrispondente alla codifica binaria $1111,1111_2$.

Se, invece, vogliamo rappresentare anche numeri negativi, dobbiamo utilizzare uno dei 4 bit assegnati alla parte intera o alla parte decimale, In entrambe i casi avremo metà numeri positivi e metà numeri negativi. Se il bit è preso dalla parte intera allora il valore massimo sarà +7,9375 ($0111,1111_2$), mentre quello minimo sarà -7,9375 ($1111,1111_2$).



Se il bit è preso dalla parte frazionaria allora il valore massimo sarà +15,875 ($01111,111_2$), mentre quello minimo sarà -15,875 ($11111,111_2$).



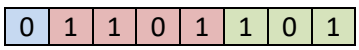
Purtroppo, questa soluzione non offre un uso ottimizzato dei bit a disposizione, ad esempio numeri interi o numeri solamente frazionari, sprecano metà dei bit a disposizione.

Esempio

Supponiamo di voler rappresentare il numero +13,625 su 8 bit con 1 bit di segno, 4 bit per la parte intera e 3 per la parte decimale:

- Segno + $\Rightarrow$ 0
- 13 $\Rightarrow$ 1101_2 perché $13 = 8 + 4 + 1 = 2^3 + 2^2 + 2^0$
- $0,625 = 0,101_2$ perché $0,625 = 0,5 + 0,125 = 2^{-1} + 2^{-3}$

Quindi il Byte sarà così riempito:



5.4 Rappresentazione in virgola mobile (IEEE 754)

Lo standard IEEE per il calcolo in **virgola mobile** (IEEE 754) (ufficialmente: IEEE Standard for Binary Floating-Point Arithmetic 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 optional.

5.4.1 Struttura di un numero in virgola mobile

Un numero in virgola mobile, secondo lo standard IEEE è rappresentato da parole di 32, 64 o 80 bit divisi in 3 parti in questo ordine:

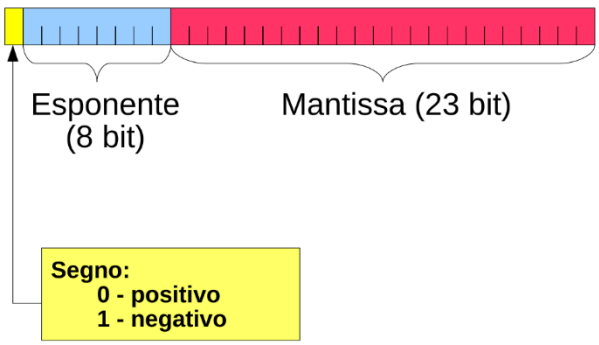
- un bit di segno **s**;
- l'esponente **e**;
- la mantissa **m**

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.

5.4.2 Numeri a precisione singola (32 bit)

Di fianco è riportata la struttura di un numero in virgola mobile a 32 bit.
Il valore del numero rappresentato è calcolabile come:

$$(-1)^S \times 2^E \times M$$



Il campo **S** specifica il segno del numero: 0 per i numeri positivi, 1 per i numeri negativi. Il campo **E** contiene l'esponente del numero in forma intera. Essendo costituito da 8 bit, permette di rappresentare 256 valori.

I valori 0 e 255 vengono riservati per funzioni speciali (descritte in seguito); gli altri permettono di rappresentare 254 valori per i numeri in forma normale, compresi tra -126 e 127, dato che questo campo deve poter rappresentare sia numeri enormi che minimi. Tuttavia, adoperando il metodo usato per la rappresentazione del segno dei numeri interi, si creerebbero problemi per il confronto tra numeri. Per risolvere questo problema, si fa uso del metodo dell'eccesso aggiungendo un bias k=127, per cui:

$$e = E + k$$

e reciprocamente

$$E = e - k$$

In questo standard, per i numeri a precisione singola, il bias è uguale a 127. In questa rappresentazione (chiamata polarizzazione), i valori dell'esponente compresi tra -126 e +127 assumono, nella scrittura all'interno del byte, i valori compresi tra 1 e 254, eliminando la necessità di un bit di segno. In fase di decodifica del numero, il bias viene sottratto per recuperare il valore originale.

I valori assunti rispettivamente dall'esponente **e** e dalla mantissa **m** determinano l'appartenenza del numero ad una di queste categorie:

- zeri;
- numeri in forma normale;
- numeri in forma denormalizzata;
- infiniti;
- NaN (Not a Number).

L'esponente distingue i numeri in modo primario, la mantissa in modo secondario.

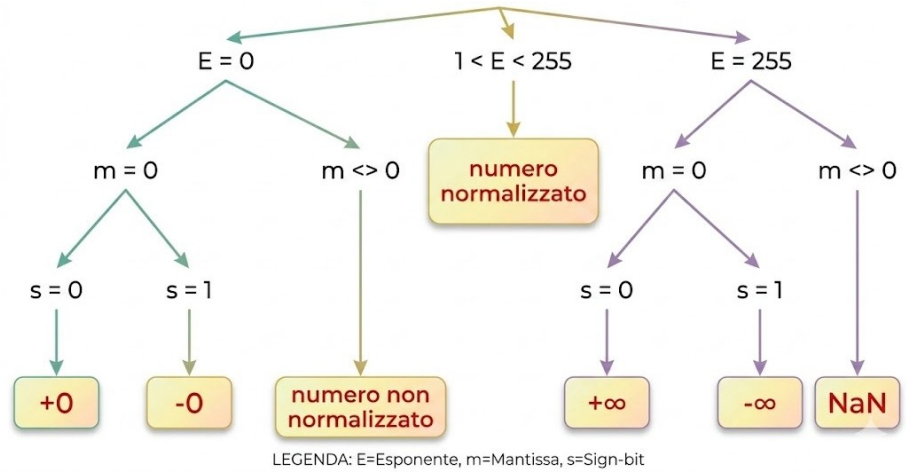


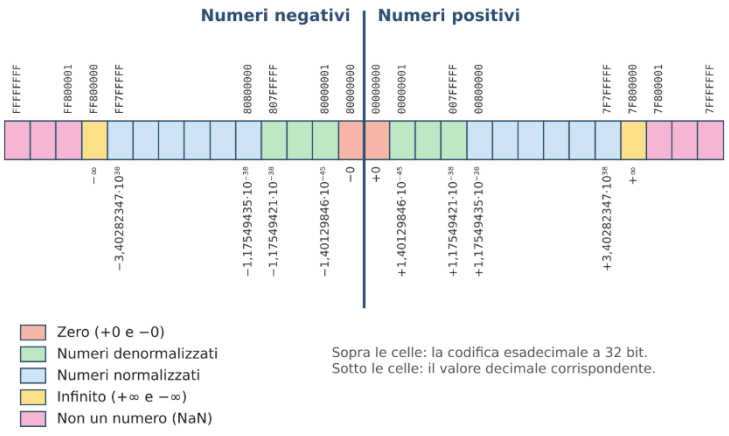
Figura 7 – Classificazione dei numeri in virgola mobile IEEE 754 (Singola Precisione)

Il campo **m** è una stringa di bit che rappresenta la sequenza di cifre dopo la virgola. Tutte le mantisse sono normalizzate in modo che il numero prima della virgola sia 1, per cui per un dato **m** il valore matematico corrispondente è:

M=1,m

In pratica, la mantissa è costituita dal numero binario 1, seguito dalla virgola e dalla parte intera del numero rappresentato, in forma binaria; la mantissa risulta così artificialmente compresa tra 1 (quando m=0) e 2 (quando m=1111111…). Quando un numero è normalizzato il primo bit della mantissa, pari a 1, viene omesso per convenienza: viene quindi chiamato **bit nascosto**, o **bit implicito**.

Con questo sistema di rappresentazione, si hanno due zeri (+0 e −0) e due infiniti (+∞ e −∞) a seconda del valore del primo bit. Inoltre, i numeri subnormali possono avere un segno e una mantissa, utili però solo per l'analisi.



5.4.3 Esempio di Codifica

Facciamo un semplice esempio: codifichiamo il numero n=−118.5 nel

sistema IEEE 754. Dobbiamo determinarne il segno, l'esponente e la mantissa. Poiché è un numero negativo, il primo bit è "1".

Poi scriviamo il numero in forma binaria senza segno: $1110110{,}1_2$.

Successivamente spostiamo la virgola verso sinistra, lasciando solo un 1 alla sua sinistra:

$$1110110{,}1 = 1{,}1101101 \cdot 2^6$$

La mantissa è la parte a destra della virgola, riempita con zeri a destra fino a riempire i 23 bit:

$$m=11011010000000000000000.$$

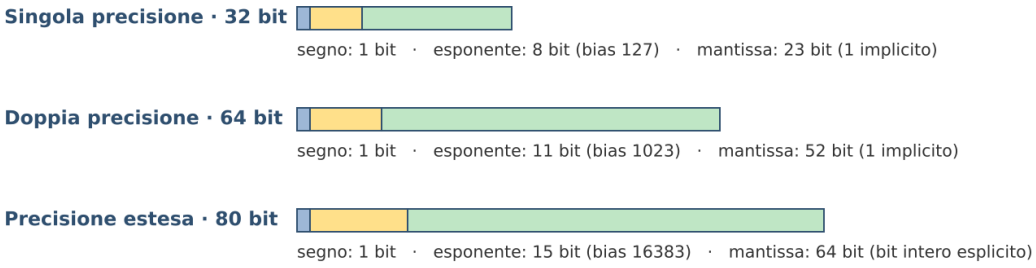
L'esponente è pari a 6, ma dobbiamo convertirlo in forma binaria e adattarlo allo standard. Per la precisione singola, dobbiamo aggiungere 127. Quindi 6 + 127 = 133. In forma binaria: 10000101.

Assemblando il tutto:

s	esponente								mantissa																					
1	1	0	0	0	0	1	0	1	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5.4.4 Formati più usati

Formato	Bit totali	Bit segno	Bit caratteristica	Bit mantissa
Singola precisione	32	1	8	23
Doppia precisione	64	1	11	52
Precisione estesa	80	1	15	64



5.4.5 Intervallo dell'esponente in base 10

Vogliamo ora capire quale è l'ordine di grandezza minimo e massimo dei numeri che si possono rappresentare con i tre formati più utilizzati.
Consideriamo il caso della singola precisione (32 bit totali):

- $1{,}00000000000000000000000 \times 2^{-126} = 2^{-126} \approx 1{,}2 \times 10^{-38}$
- $1{,}11111111111111111111111 \times 2^{127} \approx 2 \times 2^{127} = 2^{128} \approx 3{,}4 \times 10^{38}$

Per quanto riguarda il caso della doppia precisione (64 bit totali):

- $1{,}0000000000000000000000000000000000000000000000000000 \times 2^{-1022} = 2^{-1022} \approx 2{,}2 \times 10^{-308}$
- $1{,}1111111111111111111111111111111111111111111111111111 \times 2^{1023} = 2 \times 2^{1023} = 2^{1024} \approx 1{,}8 \times 10^{308}$

5.4.6 Esempio di decodifica

Consideriamo il seguente numero:

s	esponente								mantissa																					
0	1	0	0	0	0	1	1	1	1	0	0	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Allora si avrà:

- s = 0 quindi il numero è positivo
- e = 10000111_2=135, mentre E=135-127=8
- M = 1,m = 1,1001101111 da cui m=1001101111

Quindi il numero sarà:

- $1{,}1001101111 \times 2^8 = 110011011{,}11_2 = 256+128+16+8+2+1+0{,}5+0{,}25 = 411{,}75$

5.4.7 Approfondimento

Elencare i numeri rappresentabili con 1 bit di segno, 3 di esponente e 4 di mantissa.

Nota: Fattore di scala

E = 0 si usa lo stesso di E=1

E = 111 non viene considerato

Negli altri casi vale $2^{(E-3)}$

Segno	Esp	Man	Esp	Man	Fat-di-Scala	Significato	N=M*FdS	Tipo	Significato	Δ=FdS/2^4
0	000	0000	0	0	0,2500	0,0000	0,000000	0		
0	000	0001	0	1	0,2500	0,0625	0,015625	Denormalizzato	M/2^4	0,015625
0	000	0010	0	2	0,2500	0,1250	0,031250	Denormalizzato	M/2^4	0,015625
0	000	0011	0	3	0,2500	0,1875	0,046875	Denormalizzato	M/2^4	0,015625
0	000	0100	0	4	0,2500	0,2500	0,062500	Denormalizzato	M/2^4	0,015625
0	000	0101	0	5	0,2500	0,3125	0,078125	Denormalizzato	M/2^4	0,015625
0	000	0110	0	6	0,2500	0,3750	0,093750	Denormalizzato	M/2^4	0,015625
0	000	0111	0	7	0,2500	0,4375	0,109375	Denormalizzato	M/2^4	0,015625
0	000	1000	0	8	0,2500	0,5000	0,125000	Denormalizzato	M/2^4	0,015625
0	000	1001	0	9	0,2500	0,5625	0,140625	Denormalizzato	M/2^4	0,015625
0	000	1010	0	10	0,2500	0,6250	0,156250	Denormalizzato	M/2^4	0,015625
0	000	1011	0	11	0,2500	0,6875	0,171875	Denormalizzato	M/2^4	0,015625
0	000	1100	0	12	0,2500	0,7500	0,187500	Denormalizzato	M/2^4	0,015625
0	000	1101	0	13	0,2500	0,8125	0,203125	Denormalizzato	M/2^4	0,015625
0	000	1110	0	14	0,2500	0,8750	0,218750	Denormalizzato	M/2^4	0,015625
0	000	1111	0	15	0,2500	0,9375	0,234375	Denormalizzato	M/2^4	0,015625
0	001	0000	1	0	0,2500	1,0000	0,250000	Normalizzato	1+ M/2^4	0,015625
0	001	0001	1	1	0,2500	1,0625	0,265625	Normalizzato	1+ M/2^4	0,015625
0	001	0010	1	2	0,2500	1,1250	0,281250	Normalizzato	1+ M/2^4	0,015625
0	001	0011	1	3	0,2500	1,1875	0,296875	Normalizzato	1+ M/2^4	0,015625
0	001	0100	1	4	0,2500	1,2500	0,312500	Normalizzato	1+ M/2^4	0,015625
0	001	0101	1	5	0,2500	1,3125	0,328125	Normalizzato	1+ M/2^4	0,015625
0	001	0110	1	6	0,2500	1,3750	0,343750	Normalizzato	1+ M/2^4	0,015625
0	001	0111	1	7	0,2500	1,4375	0,359375	Normalizzato	1+ M/2^4	0,015625
0	001	1000	1	8	0,2500	1,5000	0,375000	Normalizzato	1+ M/2^4	0,015625
0	001	1001	1	9	0,2500	1,5625	0,390625	Normalizzato	1+ M/2^4	0,015625
0	001	1010	1	10	0,2500	1,6250	0,406250	Normalizzato	1+ M/2^4	0,015625
0	001	1011	1	11	0,2500	1,6875	0,421875	Normalizzato	1+ M/2^4	0,015625
0	001	1100	1	12	0,2500	1,7500	0,437500	Normalizzato	1+ M/2^4	0,015625
0	001	1101	1	13	0,2500	1,8125	0,453125	Normalizzato	1+ M/2^4	0,015625
0	001	1110	1	14	0,2500	1,8750	0,468750	Normalizzato	1+ M/2^4	0,015625
0	001	1111	1	15	0,2500	1,9375	0,484375	Normalizzato	1+ M/2^4	0,015625
0	010	0000	2	0	0,5000	1,0000	0,500000	Normalizzato	1+ M/2^4	0,031250
0	010	0001	2	1	0,5000	1,0625	0,531250	Normalizzato	1+ M/2^4	0,031250
0	010	0010	2	2	0,5000	1,1250	0,562500	Normalizzato	1+ M/2^4	0,031250
0	010	0011	2	3	0,5000	1,1875	0,593750	Normalizzato	1+ M/2^4	0,031250
0	010	0100	2	4	0,5000	1,2500	0,625000	Normalizzato	1+ M/2^4	0,031250
0	010	0101	2	5	0,5000	1,3125	0,656250	Normalizzato	1+ M/2^4	0,031250
0	010	0110	2	6	0,5000	1,3750	0,687500	Normalizzato	1+ M/2^4	0,031250
0	010	0111	2	7	0,5000	1,4375	0,718750	Normalizzato	1+ M/2^4	0,031250
0	010	1000	2	8	0,5000	1,5000	0,750000	Normalizzato	1+ M/2^4	0,031250

0	010	1001	2	9	0,5000	1,5625	0,781250	Normalizzato	$1+ M/2^4$	0,031250
0	010	1010	2	10	0,5000	1,6250	0,812500	Normalizzato	$1+ M/2^4$	0,031250
0	010	1011	2	11	0,5000	1,6875	0,843750	Normalizzato	$1+ M/2^4$	0,031250
0	010	1100	2	12	0,5000	1,7500	0,875000	Normalizzato	$1+ M/2^4$	0,031250
0	010	1101	2	13	0,5000	1,8125	0,906250	Normalizzato	$1+ M/2^4$	0,031250
0	010	1110	2	14	0,5000	1,8750	0,937500	Normalizzato	$1+ M/2^4$	0,031250
0	010	1111	2	15	0,5000	1,9375	0,968750	Normalizzato	$1+ M/2^4$	0,031250
0	011	0000	3	0	1,0000	1,0000	1,000000	Normalizzato	$1+ M/2^4$	0,062500
0	011	0001	3	1	1,0000	1,0625	1,062500	Normalizzato	$1+ M/2^4$	0,062500
0	011	0010	3	2	1,0000	1,1250	1,125000	Normalizzato	$1+ M/2^4$	0,062500
0	011	0011	3	3	1,0000	1,1875	1,187500	Normalizzato	$1+ M/2^4$	0,062500
0	011	0100	3	4	1,0000	1,2500	1,250000	Normalizzato	$1+ M/2^4$	0,062500
0	011	0101	3	5	1,0000	1,3125	1,312500	Normalizzato	$1+ M/2^4$	0,062500
0	011	0110	3	6	1,0000	1,3750	1,375000	Normalizzato	$1+ M/2^4$	0,062500
0	011	0111	3	7	1,0000	1,4375	1,437500	Normalizzato	$1+ M/2^4$	0,062500
0	011	1000	3	8	1,0000	1,5000	1,500000	Normalizzato	$1+ M/2^4$	0,062500
0	011	1001	3	9	1,0000	1,5625	1,562500	Normalizzato	$1+ M/2^4$	0,062500
0	011	1010	3	10	1,0000	1,6250	1,625000	Normalizzato	$1+ M/2^4$	0,062500
0	011	1011	3	11	1,0000	1,6875	1,687500	Normalizzato	$1+ M/2^4$	0,062500
0	011	1100	3	12	1,0000	1,7500	1,750000	Normalizzato	$1+ M/2^4$	0,062500
0	011	1101	3	13	1,0000	1,8125	1,812500	Normalizzato	$1+ M/2^4$	0,062500
0	011	1110	3	14	1,0000	1,8750	1,875000	Normalizzato	$1+ M/2^4$	0,062500
0	011	1111	3	15	1,0000	1,9375	1,937500	Normalizzato	$1+ M/2^4$	0,062500
0	100	0000	4	0	2,0000	1,0000	2,000000	Normalizzato	$1+ M/2^4$	0,125000
0	100	0001	4	1	2,0000	1,0625	2,125000	Normalizzato	$1+ M/2^4$	0,125000
0	100	0010	4	2	2,0000	1,1250	2,250000	Normalizzato	$1+ M/2^4$	0,125000
0	100	0011	4	3	2,0000	1,1875	2,375000	Normalizzato	$1+ M/2^4$	0,125000
0	100	0100	4	4	2,0000	1,2500	2,500000	Normalizzato	$1+ M/2^4$	0,125000
0	100	0101	4	5	2,0000	1,3125	2,625000	Normalizzato	$1+ M/2^4$	0,125000
0	100	0110	4	6	2,0000	1,3750	2,750000	Normalizzato	$1+ M/2^4$	0,125000
0	100	0111	4	7	2,0000	1,4375	2,875000	Normalizzato	$1+ M/2^4$	0,125000
0	100	1000	4	8	2,0000	1,5000	3,000000	Normalizzato	$1+ M/2^4$	0,125000
0	100	1001	4	9	2,0000	1,5625	3,125000	Normalizzato	$1+ M/2^4$	0,125000
0	100	1010	4	10	2,0000	1,6250	3,250000	Normalizzato	$1+ M/2^4$	0,125000
0	100	1011	4	11	2,0000	1,6875	3,375000	Normalizzato	$1+ M/2^4$	0,125000
0	100	1100	4	12	2,0000	1,7500	3,500000	Normalizzato	$1+ M/2^4$	0,125000
0	100	1101	4	13	2,0000	1,8125	3,625000	Normalizzato	$1+ M/2^4$	0,125000
0	100	1110	4	14	2,0000	1,8750	3,750000	Normalizzato	$1+ M/2^4$	0,125000
0	100	1111	4	15	2,0000	1,9375	3,875000	Normalizzato	$1+ M/2^4$	0,125000
0	101	0000	5	0	4,0000	1,0000	4,000000	Normalizzato	$1+ M/2^4$	0,250000
0	101	0001	5	1	4,0000	1,0625	4,250000	Normalizzato	$1+ M/2^4$	0,250000
0	101	0010	5	2	4,0000	1,1250	4,500000	Normalizzato	$1+ M/2^4$	0,250000
0	101	0011	5	3	4,0000	1,1875	4,750000	Normalizzato	$1+ M/2^4$	0,250000
0	101	0100	5	4	4,0000	1,2500	5,000000	Normalizzato	$1+ M/2^4$	0,250000
0	101	0101	5	5	4,0000	1,3125	5,250000	Normalizzato	$1+ M/2^4$	0,250000
0	101	0110	5	6	4,0000	1,3750	5,500000	Normalizzato	$1+ M/2^4$	0,250000
0	101	0111	5	7	4,0000	1,4375	5,750000	Normalizzato	$1+ M/2^4$	0,250000
0	101	1000	5	8	4,0000	1,5000	6,000000	Normalizzato	$1+ M/2^4$	0,250000
0	101	1001	5	9	4,0000	1,5625	6,250000	Normalizzato	$1+ M/2^4$	0,250000
0	101	1010	5	10	4,0000	1,6250	6,500000	Normalizzato	$1+ M/2^4$	0,250000
0	101	1011	5	11	4,0000	1,6875	6,750000	Normalizzato	$1+ M/2^4$	0,250000
0	101	1100	5	12	4,0000	1,7500	7,000000	Normalizzato	$1+ M/2^4$	0,250000
0	101	1101	5	13	4,0000	1,8125	7,250000	Normalizzato	$1+ M/2^4$	0,250000
0	101	1110	5	14	4,0000	1,8750	7,500000	Normalizzato	$1+ M/2^4$	0,250000
0	101	1111	5	15	4,0000	1,9375	7,750000	Normalizzato	$1+ M/2^4$	0,250000
0	110	0000	6	0	8,0000	1,0000	8,000000	Normalizzato	$1+ M/2^4$	0,500000
0	110	0001	6	1	8,0000	1,0625	8,500000	Normalizzato	$1+ M/2^4$	0,500000
0	110	0010	6	2	8,0000	1,1250	9,000000	Normalizzato	$1+ M/2^4$	0,500000
0	110	0011	6	3	8,0000	1,1875	9,500000	Normalizzato	$1+ M/2^4$	0,500000

0	110	0100	6	4	8,0000	1,2500	10,00000	Normalizzato	1+ M/2^4	0,500000
0	110	0101	6	5	8,0000	1,3125	10,50000	Normalizzato	1+ M/2^4	0,500000
0	110	0110	6	6	8,0000	1,3750	11,00000	Normalizzato	1+ M/2^4	0,500000
0	110	0111	6	7	8,0000	1,4375	11,50000	Normalizzato	1+ M/2^4	0,500000
0	110	1000	6	8	8,0000	1,5000	12,00000	Normalizzato	1+ M/2^4	0,500000
0	110	1001	6	9	8,0000	1,5625	12,50000	Normalizzato	1+ M/2^4	0,500000
0	110	1010	6	10	8,0000	1,6250	13,00000	Normalizzato	1+ M/2^4	0,500000
0	110	1011	6	11	8,0000	1,6875	13,50000	Normalizzato	1+ M/2^4	0,500000
0	110	1100	6	12	8,0000	1,7500	14,00000	Normalizzato	1+ M/2^4	0,500000
0	110	1101	6	13	8,0000	1,8125	14,50000	Normalizzato	1+ M/2^4	0,500000
0	110	1110	6	14	8,0000	1,8750	15,00000	Normalizzato	1+ M/2^4	0,500000
0	110	1111	6	15	8,0000	1,9375	15,50000	Normalizzato	1+ M/2^4	0,500000
0	111	0000	7	0	+infinito					
0	111	0001	7	1	NaN					
…	…	…	…	…	…					
0	111	1111	7	15	NaN					
1	111	0000	7	0	-infinito					
1	111	0001	7	1	NaN					
…	…	…	…	…	…					
1	111	1111	7	15	NaN					

6 Codifiche per il testo o come testo

6.1 La codifica ASCII

Il codice ASCII (**American Standard Code for Information Interchange**) è invece uno standard approvato dallo ANSI (**American National Standard Institute**) nel 1963 e fatto proprio dalla ISO (**International Standard Organization**) come ISO 646.

La prima versione dello standard prevedeva parole di codice lunghe 7 bit; il bit eccedente nel Byte veniva utilizzato come **bit di parità** per una blanda forma di controllo sulla correttezza dei 7 bit della codifica.

Tale bit veniva infatti posto a zero o uno in modo da far risultare pari la quantità di bit impostati al valore uno in ogni byte; in questo modo un errore di rappresentazione o di trasmissione dati di un Byte che facesse diventare dispari tale quantità, poteva essere rilevato (il controllo era però inefficace nel caso di due errori sullo stesso byte che si compensavano lasciando pari la quantità di bit con valore uno).

I simboli codificabili in ASCII ISO 646 sono 128 e si suddividono in:

- **riproducibili**, cioè che si trovano sulle tastiere e possono essere stampati a video o su carta;
- **non riproducibili**, o di controllo o non stampabili, la maggior parte utilizzati per gestire le telescriventi, apparecchiature molto usate ai tempi della definizione dello standard (comunque molti di questi simboli conservano anche oggi una notevole utilità); essi, a loro volta, si suddividono in:
  - **simboli di trasmissione**, ad esempio: STX = Start Of Text (inizio testo) o ACK = Acknowledge (conferma positiva);
  - **simboli di formato**, ad esempio: LF = Line Feed (nuova linea), FF = Form Feed, (nuova pagina), CR = Carriage Return (ritorno a capo);
  - **simboli ausiliari**, ad esempio: DEL = Delete (cancella carattere), BEL = Bell (emissione di un suono), NUL = Null (carattere nullo).

I simboli non riproducibili nella tabella dei codici ASCII sono i primi trentadue, quelli il cui codice tradotto in valore decimale è compreso tra 0 e 31.

Seguono poi, inframezzati a simboli vari e di punteggiatura, le cifre decimali, le lettere maiuscole e le lettere minuscole.

Il codice ASCII è usato da tutti i produttori diversi da IBM e anche dalla stessa IBM nel caso dei Personal Computer; in Figura 8 vediamo la tabella dei codici nella quale notiamo che le parole di codice sono lunghe otto bit: si tratta in effetti della prima metà della tabella ASCII estesa contenente gli stessi codici di quella standard ma con un bit a zero in più a sinistra.

Esiste infatti, dal 1985, una versione più moderna del codice ASCII, chiamata ASCII estesa o ISO 8859, in cui la lunghezza delle parole è di 8 bit.

I simboli codificabili diventano così 256 e questo permette di codificare correttamente le lettere degli alfabeti diversi da quello inglese, latino e swahili (che sono privi di lettere accentate) i soli per i quali è sufficiente la versione a sette bit.

C'è poi la possibilità di inserire nel codice anche molti altri simboli fra cui ad esempio quelli grafici; inoltre la perdita del bit di parità non è importante in quanto il controllo realizzato con esso non era molto sofisticato e affidabile.

L'estensione del codice è stata fatta in modo «intelligente» lasciando inalterati i primi 128 simboli (a parte l'aumento di lunghezza delle parole) in modo da assicurare la «retro-compatibilità» con la versione precedente del codice.

Binario	Dec	Carattere	Binario	Dec	Carattere	Binario	Dec	Carattere	Binario	Dec	Carattere
00000000	0	NUL – nullo	00100000	32	spazio	01000000	64	@	01100000	96	`
00000001	1	SOH – inizio intestazione	00100001	33	!	01000001	65	A	01100001	97	a
00000010	2	STX – inizio testo	00100010	34	"	01000010	66	B	01100010	98	b
00000011	3	ETX – fine testo	00100011	35	#	01000011	67	C	01100011	99	c
00000100	4	EOT – fine trasmissione	00100100	36	$	01000100	68	D	01100100	100	d
00000101	5	ENQ – richiesta	00100101	37	%	01000101	69	E	01100101	101	e
00000110	6	ACK – conferma	00100110	38	&	01000110	70	F	01100110	102	f
00000111	7	BEL – campanello	00100111	39	'	01000111	71	G	01100111	103	g
00001000	8	BS – spazio indietro	00101000	40	(	01001000	72	H	01101000	104	h
00001001	9	HT – tabulazione orizz.	00101001	41	)	01001001	73	I	01101001	105	i
00001010	10	LF – nuova riga	00101010	42	*	01001010	74	J	01101010	106	j
00001011	11	VT – tabulazione vert.	00101011	43	+	01001011	75	K	01101011	107	k
00001100	12	FF – nuova pagina	00101100	44	,	01001100	76	L	01101100	108	l
00001101	13	CR – ritorno a capo	00101101	45	-	01001101	77	M	01101101	109	m
00001110	14	SO – shift out	00101110	46	.	01001110	78	N	01101110	110	n
00001111	15	SI – shift in	00101111	47	/	01001111	79	O	01101111	111	o
00010000	16	DLE – escape collegamento	00110000	48	0	01010000	80	P	01110000	112	p
00010001	17	DC1 – controllo disp. 1	00110001	49	1	01010001	81	Q	01110001	113	q
00010010	18	DC2 – controllo disp. 2	00110010	50	2	01010010	82	R	01110010	114	r
00010011	19	DC3 – controllo disp. 3	00110011	51	3	01010011	83	S	01110011	115	s
00010100	20	DC4 – controllo disp. 4	00110100	52	4	01010100	84	T	01110100	116	t
00010101	21	NAK – conferma negativa	00110101	53	5	01010101	85	U	01110101	117	u
00010110	22	SYN – attesa sincrona	00110110	54	6	01010110	86	V	01110110	118	v
00010111	23	ETB – fine blocco	00110111	55	7	01010111	87	W	01110111	119	w
00011000	24	CAN – annulla	00111000	56	8	01011000	88	X	01111000	120	x
00011001	25	EM – fine supporto	00111001	57	9	01011001	89	Y	01111001	121	y
00011010	26	SUB – sostituzione	00111010	58	:	01011010	90	Z	01111010	122	z
00011011	27	ESC – escape	00111011	59	;	01011011	91	[	01111011	123	{
00011100	28	FS – separatore di file	00111100	60	<	01011100	92	\	01111100	124	|
00011101	29	GS – separatore di gruppo	00111101	61	=	01011101	93	]	01111101	125	}
00011110	30	RS – separatore di record	00111110	62	>	01011110	94	^	01111110	126	~
00011111	31	US – separatore di unità	00111111	63	?	01011111	95	_	01111111	127	DEL – cancella

Figura 8 – Tabella ASCII – Primi 128 caratteri

Sono stati poi creati un certo numero di sotto-standard, che si differenziano per gli ultimi 96 degli altri 128 simboli (i primi 32 sono ulteriori caratteri di controllo comuni a tutte le varianti dello standard).

Ognuno dei sotto-standard è dedicato ad una particolare lingua o a un gruppo di lingue; abbiamo quindi:

- ISO 8859-1 (Latin-1 Western European): per le lingue dell'Europa Occidentale e della Scandinavia (quindi anche delle Americhe, dell'Oceania dell'India e altre ex colonie europee https://www.ascii-code.com/it);
- ISO 8859-2 (Latin-2 Central European): per le lingue dell'Europa centro orientale basate sull'alfabeto latino;
- ISO 8859-3 (Latin-3 South European): per maltese, turco e esperanto;
- ISO 8859-4 (Latin-4 North European): lingue dei paesi baltici e della Groenlandia;
- ISO 8859-5 (Latin/Cyrillic): lingue basate su alfabeto cirillico;
- ISO 8859-6 (Latin/Arabic): arabo;
- ISO 8859-7 (Latin/Greek): greco moderno;
- ISO 8859-8 (Latin/hebrew): ebraico moderno;
- ISO 8859-9 (Latin-5 Turkish): come Latin-1 con l'islandese rimpiazzato dal turco;
- ISO 8859-10 (Latin-6 Nordic): modifica del Latin-4 per i linguaggi nordici;

- ISO 8859-11 (Latin/Thai): linguaggio thai;
- ISO 8859-13 (Latin-7 Baltic Rim): aggiunta di caratteri mancanti in Latin-4 e Latin-6;
- ISO 8859-14 (Latin-8 Celtic): lingua celtica;
- ISO 8859-15 (Latin-9): revisione del Latin-1 con l'introduzione, tra l'altro del simbolo dell'Euro;
- ISO 8859-16 (Latin-10 South-Eastern European): definito per alcune lingue tra cui italiano, sloveno, francese, tedesco con il simbolo di valuta sostituito da quello dell'Euro.

6.1.1 Esempio di codifica e decodifica

Vogliamo memorizzare la stringa di caratteri "Esempio 3bis" secondo la codifica ASCII a 8 bit. Come prima cosa trasformiamo ogni carattere nel numero decimale corrispondente.

E=69, s=115, e=101, m=109, p=112, i=105, o=111, *spazio*=32, 3=51, b=98, i=105, s=115

A questo punto non rimane che inserire le codifiche binarie su 12 Byte di memoria.

0	1	0	0	0	1	0	1	E
0	1	1	1	0	0	1	1	s
0	1	1	0	0	1	0	1	e
0	1	1	0	1	1	0	1	m
0	1	1	1	0	0	0	0	p
0	1	1	0	1	0	0	1	i
0	1	1	0	1	1	1	1	o
0	0	1	0	0	0	0	0	
0	0	1	1	0	0	1	1	3
0	1	1	1	0	0	1	1	b
0	1	1	0	1	0	0	1	i
0	1	1	1	0	0	1	1	s

La decodifica prevede il percorso inverso, pertanto da ogni Byte, bisognerà risalire al valore decimale corrispondente e da esso, attraverso la tabella ASCII, al carattere corrispondente. La sequenza di caratteri così individuata sarà la stringa cercata.

6.2 Codifica Base64

Base64 è un sistema di codifica che consente la traduzione di dati binari in stringhe di testo ASCII, rappresentando i dati sulla base di 64 caratteri ASCII diversi.

L'algoritmo che effettua la conversione suddivide il file in gruppi da 6 bit, i quali possono quindi contenere valori da 0 a 63. Ogni possibile valore viene convertito in un carattere ASCII secondo la seguente tabella:

Valore	ASCII
0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	I
9	J
10	K
11	L
12	M
13	N
14	O
15	P

Valore	ASCII
16	Q
17	R
18	S
19	T
20	U
21	V
22	W
23	X
24	Y
25	Z
26	a
27	b
28	c
29	d
30	e
31	f

Valore	ASCII
32	g
33	h
34	i
35	j
36	k
37	l
38	m
39	n
40	o
41	p
42	q
43	r
44	s
45	t
46	u
47	v

Valore	ASCII
48	w
49	x
50	y
51	z
52	0
53	1
54	2
55	3
56	4
57	5
58	6
59	7
60	8
61	9
62	+
63	/

Il set di 64 caratteri scelto per rappresentare i 64 valori (0-63) varia a seconda dell'implementazione scelta. L'obbiettivo solitamente è di scegliere 64 caratteri che siano comuni alla maggior parte delle codifiche e che siano anche stampabili. Le altre varianti, solitamente derivate dal Base64, condividono queste proprietà ma differiscono nella scelta degli ultimi due caratteri; per esempio la variante URL e file name safe (RFC 4648/Base64URL), usa "-" e "_"

Esempio di codifica di 3 caratteri ASCII

Dati 3 byte che contengono i valori A, B, C, partendo dalla loro rappresentazione binaria, applichiamo l'algoritmo:

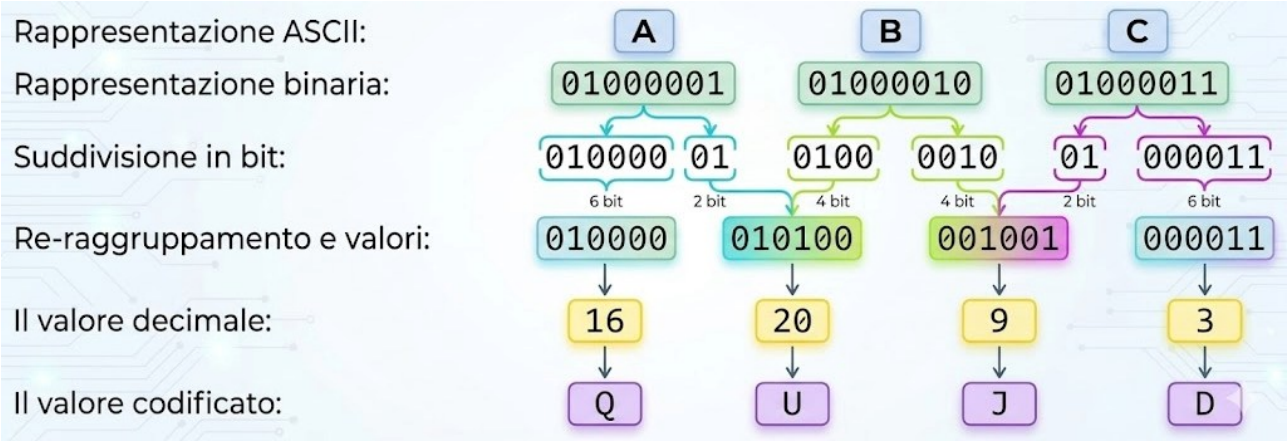


Figura 9 – Processo di conversione da ASCII a Base64

6.3 I codici UCS-2 e UTF-8

Lo standard ISO 8859 (ASCII a 8 bit), con le sue varianti, copre le necessità di molte delle lingue esistenti; rimangono però fuori tutti quei linguaggi che non sono basati su un alfabeto ma su ideogrammi ognuno dei quali può corrispondere ad una parola o addirittura ad una frase.

Per indicare questi linguaggi si usa l'acronimo **CJK** (Chinese, Japanese, Korean) dai nomi delle lingue più rappresentative in questo ambito.

La soluzione probabilmente definitiva ai problemi di codifica si ha con la creazione, a partire dal 1991, dell'insieme di caratteri **UCS** (Universal Character Set), standard ISO 10646, meglio noto come **Unicode**.

Con questo insieme di caratteri si associa ad ogni carattere o simbolo di ogni lingua del mondo un numero intero detto code point o punto di codifica.

L'insieme è in continua evoluzione e attualmente conta molte decine di migliaia di codici: sono considerati tutti gli alfabeti conosciuti, anche quelli «inventati» come il Klingon della serie televisiva Star Trek.

I punti di codifica si indicano di solito con la notazione **U+xxxx**, dove «xxxx» identifica il valore esadecimale del codice del mcarattere: ad esempio il simbolo «marchio registrato» ® ha codice U+00AE e «copyright» © il codice U+00A9. (Tabella Unicode https://www.brescianet.com/appunti/vari/unicode.htm)

Grazie ad Unicode vengono definite anche le regole di composizione in modo da formare nuovi simboli partendo da due già esistenti (ad esempio õ da «o» e «~»).

Le codifiche più importanti e utilizzate sono:

- **UCS-2** che usa 2 Byte per codificare ogni simbolo ed è utilizzata dal 1995 nei **sistemi operativi della Microsoft e dal linguaggio di programmazione Java**; non è pienamente retrocompatibile con il codice ASCII perché, sebbene i primi 128 simboli siano gli stessi, la codifica di **ogni simbolo occupa il doppio dei bit (16 bit)**;

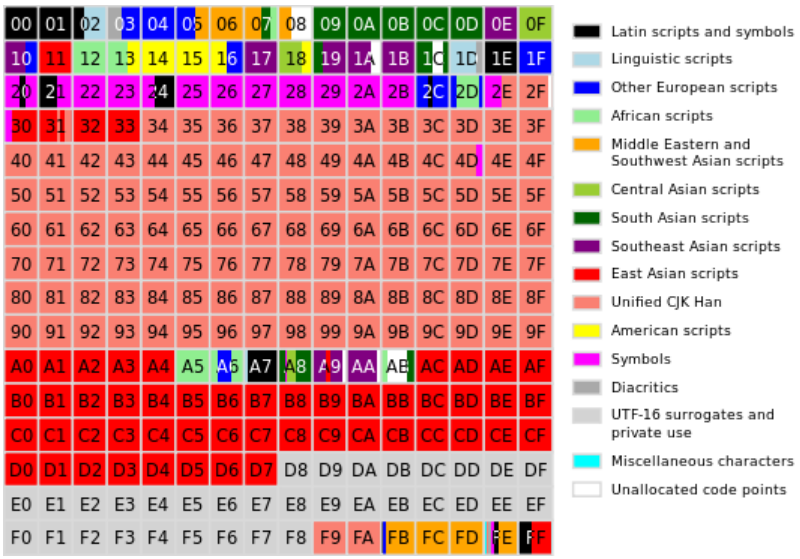


Figura 10 - Basic Multilingual Plane (Unicode a 16 bit) (immagine: Drmccreedy, Wikimedia Commons, pubblico dominio)

- **UTF-8 (Unicode Transformation Format 8)**, utilizzata nei **sistemi Unix e Linux** e riconosciuta come standard per le comunicazioni in Internet; è retro-compatibile con ASCII perché i primi 128 simboli, oltre ad essere gli stessi, sono codificati con parole di 8 bit; gli altri codici sono più lunghi (fino a 6 Byte) e quindi siamo in presenza di una codifica a **lunghezza variabile**, più complessa da gestire.

  UTF-8 usa da 1 a 4 Byte per rappresentare un carattere Unicode. Ad esempio, un solo Byte è necessario per rappresentare i 128 caratteri dell'alfabeto ASCII, corrispondenti alle posizioni Unicode da U+0000 a U+007F.
  Quattro Byte possono sembrare troppi per un solo carattere; tuttavia questo è richiesto solo per i caratteri che stanno fuori dal Basic Multilingual Plane, generalmente molto rari.

In UTF-8 i **primi 128 simboli** sono codificati con 1 Byte; quelli da 128 a 2047 invece con 2 Byte (e sono i simboli sufficienti a coprire tutte le lingue occidentali) mentre i simboli da 2048 a 65535 occupano 3 Byte (ideogrammi e altri simboli vari). Esiste anche la possibilità di utilizzare 4 Byte che permettono di avere codifiche da 65536 a 2097152, ma queste codifiche sono utilizzate per simboli che non fanno parte del Basic Multilingual Plane.

Intervallo Unicode esadecimale	UTF-8 binario	N° codifiche	Note
0x000000-0x00007F	0XXXXXXX	2^7=128	Caratteri equivalenti al codice ASCII; i byte iniziano con 0 e da soli indicano un carattere.
0x000080-0x0007FF	110XXXXX 10XXXXXX	2^{11}=2048	il primo byte inizia per 110 o 1110, il successivo(i) con 10 e devono essere concatenati per formare un carattere
0x000800-0x00FFFF	1110XXXX 10XXXXXX 10XXXXXX	2^{16}=65536	
0x010000-0x10FFFF	11110XXX 10XXXXXX 10XXXXXX 10XXXXXX	2^{21}=2097152	Confronto tra UTF-16 e UTF-8: UTF-16 richiede l'uso di coppie surrogate: viene sottratto il valore esadecimale 0x10000, in modo che la sequenza dei bit non coincida con quella usata da UTF-8

6.4 Allegato: ISO 8859-16 (Latin-10 South-Eastern European)

ISO/IEC 8859-16

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_																
1_																
2_	SP 0020 32	! 0021 33	" 0022 34	# 0023 35	$ 0024 36	% 0025 37	& 0026 38	' 0027 39	(0028 40	) 0029 41	* 002A 42	+ 002B 43	, 002C 44	- 002D 45	. 002E 46	/ 002F 47
3_	0 0030 48	1 0031 49	2 0032 50	3 0033 51	4 0034 52	5 0035 53	6 0036 54	7 0037 55	8 0038 56	9 0039 57	: 003A 58	; 003B 59	< 003C 60	= 003D 61	> 003E 62	? 003F 63
4_	@ 0040 64	A 0041 65	B 0042 66	C 0043 67	D 0044 68	E 0045 69	F 0046 70	G 0047 71	H 0048 72	I 0049 73	J 004A 74	K 004B 75	L 004C 76	M 004D 77	N 004E 78	O 004F 79
5_	P 0050 80	Q 0051 81	R 0052 82	S 0053 83	T 0054 84	U 0055 85	V 0056 86	W 0057 87	X 0058 88	Y 0059 89	Z 005A 90	[005B 91	\ 005C 92	] 005D 93	^ 005E 94	_ 005F 95
6_	` 0060 96	a 0061 97	b 0062 98	c 0063 99	d 0064 100	e 0065 101	f 0066 102	g 0067 103	h 0068 104	i 0069 105	j 006A 106	k 006B 107	l 006C 108	m 006D 109	n 006E 110	o 006F 111
7_	p 0070 112	q 0071 113	r 0072 114	s 0073 115	t 0074 116	u 0075 117	v 0076 118	w 0077 119	x 0078 120	y 0079 121	z 007A 122	{ 007B 123	| 007C 124	} 007D 125	~ 007E 126	
8_																
9_																
A_	NBSP 00A0 160	Ą 0104 161	ą 0105 162	Ł 0141 163	€ 20AC 164	„ 201E 165	Š 0160 166	§ 00A7 167	š 0161 168	© 00A9 169	Ș 0218 170	« 00AB 171	Ź 0179 172	SHY 00AD 173	ź 017A 174	Ż 017B 175

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
B_	° 00B0 176	± 00B1 177	Č 010C 178	ł 0142 179	Ž 017D 180	” 201D 181	¶ 00B6 182	· 00B7 183	ž 017E 184	č 010D 185	ș 0219 186	» 00BB 187	Œ 0152 188	œ 0153 189	Ÿ 0178 190	ż 017C 191
C_	À 00C0 192	Á 00C1 193	Â 00C2 194	Ă 0102 195	Ä 00C4 196	Ć 0106 197	Æ 00C6 198	Ç 00C7 199	È 00C8 200	É 00C9 201	Ê 00CA 202	Ë 00CB 203	Ì 00CC 204	Í 00CD 205	Î 00CE 206	Ï 00CF 207
D_	Đ 0110 208	Ń 0143 209	Ò 00D2 210	Ó 00D3 211	Ô 00D4 212	Ő 0150 213	Ö 00D6 214	Ś 015A 215	Ű 0170 216	Ù 00D9 217	Ú 00DA 218	Û 00DB 219	Ü 00DC 220	Ę 0118 221	Ț 021A 222	ß 00DF 223
E_	à 00E0 224	á 00E1 225	â 00E2 226	ă 0103 227	ä 00E4 228	ć 0107 229	æ 00E6 230	ç 00E7 231	è 00E8 232	é 00E9 233	ê 00EA 234	ë 00EB 235	ì 00EC 236	í 00ED 237	î 00EE 238	ï 00EF 239
F_	đ 0111 240	ń 0144 241	ò 00F2 242	ó 00F3 243	ô 00F4 244	ő 0151 245	ö 00F6 246	ś 015B 247	ű 0171 248	ù 00F9 249	ú 00FA 250	û 00FB 251	ü 00FC 252	ę 0119 253	ț 021B 254	ÿ 00FF 255

Figura 11 - Differenze (bordate tra ISO 8859 e ISO 8859 - 16

7 Sitografia

- https://it.wikipedia.org/wiki/Bit
- https://it.wikipedia.org/wiki/Byte
- http://www.dacrema.com/Informatica/codifica_dati.htm
- http://www.dacrema.com/Informatica/codifica_binaria.htm
- https://it.wikipedia.org/wiki/Sistema_di_numerazione_posizionale
- https://it.wikipedia.org/wiki/Sistema_numerico_binario
- https://it.wikipedia.org/wiki/Codice_Gray
- https://it.wikipedia.org/wiki/Numero_intero_(informatica)
- https://it.wikipedia.org/wiki/Rappresentazione_in_segno_e_modulo
- https://it.wikipedia.org/wiki/Complemento_a_due
- https://it.wikipedia.org/wiki/Sistema_numerico_ottale
- https://it.wikipedia.org/wiki/Sistema_numerico_decimale
- https://it.wikipedia.org/wiki/Sistema_numerico_esadecimale
- http://www.istitutomontani.it/appunti/150/IEEEE754.pdf
- http://scuola.linux.it/docs/altre_scuole/planck/rappr-dati/codifica_binaria_dell_x0027_informazione.html
- https://it.wikipedia.org/wiki/Base64#:~:text=Base64%20è%20un%20sistema%20di,i%20dati%20nel%20formato%20ASCII