In una delle molteplici possibili definizioni di *informazione,* questa viene fatta corrispondere a qualunque elemento, in grado di essere rappresentato e comunicato, che consenta di fornire o aumentare la conoscenza su cose, fatti, ecc., attraverso meccanismi di tipo logico e razionale, ovvero anche, semplicemente, di dare coscienza dell'esistenza di tali cose o fatti attraverso la sola percezione. L'informazione si presenta tipicamente come flusso di *dati*, definibili quindi come *elementi di informazione* opportunamente strutturati in funzione di ciò che rappresentano.
Alla base di qualunque processo di memorizzazione, elaborazione e trasmissione dell'informazione vi è quindi la necessità di *rappresentare* l'informazione stessa utilizzando insiemi di regole (*codici*) basati su *simboli* di cui coloro che dovranno condividerla devono essere in grado di comprendere il significato.
In natura, l'informazione si presenta, sia a livello percettivo che di comunicazione, in forma *analogica*. Ciò che fornisce informazione (il suono, la luce, il tatto) è basato sulla misurazione, attraverso opportuni strumenti o attraverso i sensi, della variazione *continua* nel tempo di una grandezza che noi consideriamo o percepiamo come *proporzionale* ('analoga') al fenomeno che essa costituisce o rappresenta.
Proprietà principale di un fenomeno continuo è la possibilità di assumere una infinità di valori all'interno di un qualsivoglia intervallo, per quanto piccolo: in un intervallo compreso fra 0 e 100 metri è possibile immaginare infinite distanze intermedie, esattamente come fra 0 e 1000 km. Inoltre, un fenomeno continuo che si evolve nel tempo passando da una certa intensità ad un'altra, assume, anche se per istanti di durata infinitesimale, tutti i possibili (infiniti) valori compresi fra il valore iniziale e quello finale.
Al contrario, una grandezza *discreta* può assumere solo un numero limitato di valori e la sua variazione si realizza come successione di un numero finito di incrementi (decrementi) di ampiezza finita e, tipicamente, uguale a, o multipla di, una quantità prefissata.
La rappresentazione *numerica* o *digitale* (dall'inglese *digit*, cifra) dell'informazione è una rappresentazione di tipo discreto. Ogni fenomeno di tipo quantitativo rappresentato in forma digitale è un'approssimazione della realtà (che è tipicamente continua e, quindi, analogica) sotto forma di *campioni.* I valori dei campioni approssimano la misura esatta dell'intensità che il fenomeno che rappresentano assume in corrispondenza di un insieme limitato di punti (istanti) di misura, normalmente fra loro equidistanti (nel tempo, nello spazio, o all'interno di un qualche altro ordinamento). Se il numero di campioni utilizzati per rappresentare una quantità finita di informazione è sufficientemente grande, è possibile, a partire da tale rappresentazione approssimata, ricostruire il fenomeno continuo che essi rappresentano con una precisione arbitraria.
La rappresentazione digitale di informazioni di tipo quantitativo è quindi costituita da un insieme limitato e ordinato di sequenze di simboli elementari (cifre). E' quindi naturale associare a tali sequenze valori numerici interi e positivi (il "valore" dei campioni in cui l'informazione analogica viene scomposta), anche se, come vedremo, tale rappresentazione è utilizzata nei moderni calcolatori in un contesto molto più ampio della sola rappresentazione di numeri o di concetti di tipo quantitativo.

Il calcolatore (digitale) nasce, in primo luogo, come strumento per fare operazioni aritmetiche. I calcolatori, in effetti, elaborano dati (costituiti, come detto, da sequenze di simboli elementari) che possono essere interpretati come numeri, compiendo operazioni meccaniche che possono essere assimilate a calcoli. Tuttavia, in funzione del *contesto* in cui operano, i calcolatori sono in grado di attribuire a tali sequenze significati diversi e più ampi della semplice rappresentazione di quantità numeriche. Questo fa sì che i calcolatori siano strumenti multimediali, utilizzabili, ad esempio, anche per elaborare e rappresentare suoni o immagini o, come nel caso dell'Intelligenza Artificiale, per rappresentare ed elaborare concetti astratti, di tipo simbolico.
Il calcolatore può quindi oggi essere considerato uno strumento universale per l'elaborazione dell'informazione. Utilizzando un calcolatore è virtualmente possibile elaborare ogni possibile forma di informazione sfruttando una stessa rappresentazione: la rappresentazione digitale.

Nel seguito verranno esaminati i diversi tipi di dato e il modo in cui essi sono rappresentati all'interno del calcolatore.

Rappresentazione dei numeri La necessità di rappresentare numeri nasce in parallelo alla necessità di comunicare, utilizzando un linguaggio in grado non solo di esprimere concetti qualitativi ma anche concetti che esprimano quantità. Quindi, come è stato necessario creare un alfabeto ed i relativi simboli che lo compongono per comporre parole e formare discorsi e, quindi, esprimere concetti, così anche per i numeri è stato necessario trovare una rappresentazione basata su simboli. Analogamente a quanto accade con le lettere dell'alfabeto nella composizione di parole e di frasi, una volta che siano composti secondo opportune regole, tali simboli permettono di comunicare concetti di tipo *quantitativo*.

Una sequenza (*stringa*) di *n* simboli tratti da un insieme (*alfabeto*) di *b* simboli diversi, può assumere b^n configurazioni diverse. Nella numerazione in base 10, che ci è familiare, si usano sequenze di simboli che comprendono le 10 cifre che vanno da 0 a 9. Quindi, nel caso della numerazione in base 10, con 4 cifre decimali si possono rappresentare 10000 (10^4) numeri diversi, che associamo in modo naturale ai valori interi da 0 (0000) a 9999. L'associazione fra una sequenza di simboli e il corrispondente valore numerico da esso rappresentato avviene attraverso una codifica, universalmente utilizzata, di tipo *posizionale*. Questo significa che il significato attribuito alla stringa si ricava considerando sia il valore dei singoli simboli che la posizione che questi assumono all'interno della stringa. Se, ad esempio, si considera la stringa 3457, essa significa:

$7 * 10^0 + 5 * 10^1 + 4 * 10^2 + 3 * 10^3$.

Quindi, in generale, se abbiamo una stringa ... x y z di simboli *in base b*, cioè appartenenti ad un alfabeto che comprende b simboli, la quantità da essa rappresentata sarà $z * b^0 + y * b^1 + x * b^2$ … e così via.
La cifra che si trova più a sinistra nel numero è detta *cifra più significativa* poiché è quella che, nella decodifica del numero, viene moltiplicata per la potenza della base *b* di ordine più alto; quella più a destra è, per lo stesso motivo, detta *cifra meno significativa*.
Quindi, in una notazione di tipo posizionale, per rappresentare le quantità è necessario per prima cosa specificare la base che si vuole utilizzare. Nella vita di tutti i giorni siamo abituati ad utilizzare la notazione *in base 10* o *decimale* (b=10) che trova una radice storica nel fatto che abbiamo dieci dita, e quindi il metodo più primitivo e intuitivo per comunicare concetti quantitativi è stato, da sempre, rappresentare con le dita le quantità.
Se l'uomo ha la capacità di rappresentare dieci quantità diverse con le mani, e quindi trova naturale l'uso della base 10, il calcolatore è costituito da circuiti in grado di assumere due soli stati: è quindi naturale utilizzare, per descrivere i fenomeni elettrici che avvengono all'interno dei calcolatori, ma anche per progettarne la realizzazione fisica, una rappresentazione *binaria* (cioè in base 2), i cui simboli sono soltanto 0 e 1, che possono essere interpretati come 'livello basso o alto', 'interruttore aperto o chiuso', o come i concetti di 'falso o vero', se consideriamo la logica booleana con la quale, come si vedrà nel seguito, è possibile rappresentare concetti e processi di tipo logico con un formalismo di tipo algebrico.

Conversione di numeri da una base ad un'altra

Dato un numero rappresentato in una certa base, può essere utile operare una conversione della sua rappresentazione dalla base utilizzata a una base diversa. Per convertire in base N un numero espresso in un'altra base[1], ad esempio in base 10, esiste un metodo molto semplice, detto *metodo delle divisioni successive*, che viene illustrato nel seguito.

[1] Quando si trattano grandezze che possono essere espresse in basi differenti, o comunque non è immediatamente evidente la base che si sta utilizzando per la rappresentazione di un numero, si usa di solito una notazione in cui la sequenza di cifre che lo rappresenta appare fra parentesi e la base appare come pedice. Quindi, in generale, un numero xyzt espresso in base b si indica convenzionalmente con $(xyzt)_b$. Ad esempio, $(1157)_{10}$ significa che la sequenza di cifre 1157 va interpretata in base 10.

Metodo delle divisioni successive

Passo base:
Si divide il numero da convertire (ad es. 1157) per la base nella quale lo si vuole convertire (ad es. 2, se si vuole convertire un numero in binario) e si annota il resto della divisione:

1157 : 2 = 578 (con resto 1);

successivamente si procede allo stesso modo, dividendo il quoziente ancora per 2, (ottenendo, nel nostro caso 289) e annotando il resto (nel nostro caso, 0), etc.... fino a quando non si ottiene un quoziente nullo.
La codifica nella nuova base del numero di partenza sarà costituita dalla successione dei resti a partire dall'ultimo resto ottenuto e risalendo fino al primo (v. Esempio 1).
In questo modo (dividendo cioè per la base del sistema verso cui stiamo convertendo e considerando la successione dei resti), è possibile effettuare la conversione di un numero nella corrispondente rappresentazione *in qualunque altra base*.
E' possibile verificare che, se si applica il metodo utilizzando come 'nuova base' la stessa base in cui il numero è rappresentato (cioè si "converte" da una base verso la stessa base) la rappresentazione resta, ovviamente, invariata.
Un'osservazione immediata che si può fare se si confrontano le rappresentazioni di una stessa quantità in più basi diverse è che più piccola è la base, più lunga e meno leggibile diventa la rappresentazione della quantità stessa; la rappresentazione decimale ha trovato un uso universale, al di là della derivazione 'antropologica' legata al numero di dita che possediamo, anche perché consente di ottenere una rappresentazione dei numeri sufficientemente compatta, a partire da un insieme sufficientemente limitato di simboli.

METODO DELLE DIVISIONI SUCCESSIVE

Esempio: conversione di 1157 da base 10 a base 2

1157:2 = 578 (resto **1**)
578:2 = 289 (resto **0**)
289:2 = 144 (resto **1**)
144:2 = 72 (resto **0**)
72:2 = 36 (resto **0**)
36:2 = 18 (resto **0**)
18:2 = 9 (resto **0**)
9:2 = 4 (resto **1**)
4:2 = 2 (resto **0**)
2:2 = 1 (resto **0**)
1:2 = 0 (resto **1**)

La rappresentazione binaria (successione dei resti) di 1157 è 10010000101.

VERIFICA:

$10010000101 = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 0 \times 2^6 + 1 \times 2^7 + 0 \times 2^8 + 0 \times 2^9 + 1 \times 2^{10} =$

$= 1 + 4 + 128 + 1024 = 1157$

Esempio 1

Unità di misura binarie

Nella notazione in base 2 (*notazione binaria*) una singola cifra si chiama *bit*, che è la contrazione di BINARY DIGIT che, in inglese, non significa altro che *cifra binaria*. Con un bit si possono rappresentare soltanto due valori: 0 o l. Quindi la rappresentazione di una stessa quantità in binario richiede un maggior numero di cifre rispetto alla rappresentazione in base 10 della stessa quantità.
All'interno del calcolatore i dati sono organizzati in sequenze di bit che vengono, anche fisicamente, elaborate a blocchi di 8 bit o di dimensione pari a multipli di 8 bit.
Poiché la rappresentazione binaria, come quella decimale, è una rappresentazione posizionale, essa ne segue le stesse regole. Quindi il bit all'estrema sinistra di un numero binario è detto *bit più significativo* (o *MSB* = Most Significant Bit), poiché viene moltiplicato per la potenza di 2 di ordine più elevato; quello più a destra è detto, analogamente, *bit meno significativo* (*LSB* = Least Significant Bit).
Con i progressi nel campo della microelettronica, i circuiti (microprocessori) che eseguono le operazioni all'interno di un calcolatore sono stati in grado di gestire contemporaneamente un numero sempre più elevato di bit: nelle prime generazioni dei personal computer il blocco di dati normalmente elaborati dal calcolatore in una singola operazione era una stringa di 8 bit, cioè un *byte*. Attualmente esistono calcolatori con microprocessori che possono elaborare, con una singola operazione, quantità rappresentate da 16, 32 o 64 bit.
A partire dalla definizione di *byte* come stringa di 8 bit, è possibile definire le unità di misura utilizzate in campo informatico, come multipli del byte. Un *Kilobyte* (KB) è uguale a circa 1000 byte; in realtà sono 1024 byte perché 1024 (2^{10}) è la potenza di 2 più vicina a 1000, e viene usata come multiplo convenzionale quando si opera in base 2. A sua volta 1 *Megabyte* (MB) equivale a 1024 KByte (cioè 1024x1024 byte). Quindi non è esattamente 1 milione di byte ma un numero lievemente superiore. 1 *Gigabyte* (GB) sarà a sua volta 1024 Mbyte, cioè 2^{10} MByte, ovvero 2^{20} KByte, ovvero 2^{30} byte, cioè poco più di un miliardo di byte.

Altre rappresentazioni utilizzate in informatica

E' stato evidenziato in precedenza come, utilizzando la codifica binaria, si ottenga una rappresentazione estremamente lunga e poco leggibile delle quantità numeriche. Questo ha portato ad utilizzare, per esigenze di compattezza di notazione, rappresentazioni in basi diverse da 2, ma che fossero comunque potenze di 2: sono state utilizzate rappresentazioni di tipo *ottale* (b=8; simboli da 0 a 7) e *esadecimale* (base 16) che utilizza i dieci simboli che vanno da 0 a 9, cui si aggiungono, per raggiungere un totale di 16, le prime sei lettere dell'alfabeto che sono a, b, c, d, e, f, corrispondenti, rispettivamente, ai valori decimali 10, 11, 12, 13, 14 e 15.

Se una base b è potenza di 2 (cioè $b=2^n$), la rappresentazione binaria di una delle cifre che la compongono corrisponde esattamente a un gruppo di n cifre. Pertanto, un modo molto semplice per convertire un numero binario in un numero in una base che sia potenza di 2 consiste nel raggruppare il numero binario che si vuole convertire, a partire dal LSB, cioè da destra, in sequenze di n bit (quindi di 3 bit se si sta lavorando in ottale ($2^3=8$), di 4 bit ($2^4=16$) se si sta lavorando in esadecimale, ecc.) per poi convertire tali sequenze nella cifra corrispondente. Ad esempio, se si suddivide un numero binario in gruppi di 3 bit (con cui si possono rappresentare 8 simboli, vale a dire le cifre da 0 a 7), a partire dal bit meno significativo, e ad esse si sostituisce il corrispondente valore, si ottiene immediatamente la conversione del numero binario nella corrispondente rappresentazione ottale.
Ad esempio, la rappresentazione ottale del numero decimale 1157 (in binario 10 010 000 101) è 2205, che corrisponde a $5*8^0 + 0*8^1 + 2*8^2 + 2*8^3$ (v. Esempio 2).
Se, invece, si vuole effettuare la conversione in esadecimale si devono considerare gruppi di bit in grado di rappresentare 16 valori diversi, quindi ogni gruppo di 4 cifre binarie, a partire dal LSB, verrà messo in corrispondenza con la cifra esadecimale che rappresenta il valore espresso da tale gruppo di bit. In questo caso $(1157)_{10} = (100\ 1000\ 0101)_2 = (485)_{16}$.
Si noti come, in questo modo, un byte (una sequenza di 8 bit) sia rappresentato da due sole cifre esadecimali.

Siccome, normalmente, la quantità di dati elaborati contemporaneamente dal calcolatore è un multiplo pari di 8 bit, talvolta risulta pratico utilizzare la rappresentazione esadecimale per rappresentare in modo compatto le sequenze binarie utilizzate all'interno del calcolatore.

CONVERSIONE ESADECIMALE-BINARIO-DECIMALE

Cifra Esadecimale	Numero Binario	Decimale Cifra OOOX	Decimale Cifra OOXO	Decimale Cifra OXOO	Decimale Cifra XOOO
0	0000	0	0	0	0
1	0001	1	16	256	4096
2	0010	2	32	512	8192
3	0011	3	48	768	12288
4	0100	4	64	1024	16384
5	0101	5	80	1280	20480
6	0110	6	96	1536	24576
7	0111	7	112	1792	28672
8	1000	8	128	2048	32768
9	1001	9	144	2304	36864
A	1010	10	160	2560	40960
B	1011	11	176	2816	45056
C	1100	12	192	3072	49152
D	1101	13	208	3328	53248
E	1110	14	224	3584	57344
F	1111	15	240	3840	61440

Tabella di conversione binario/decimale/esadecimale

Conversione binario - ottale
Conversione binario - esadecimale

$(1221)_{10} = (10011000101)_2$

Binario - ottale

$(10\ 011\ 000\ 101)_2 = (2\ 3\ 0\ 5)_8$
2 3 0 5

ogni gruppo di 3 bit corrisponde ad una cifra ottale

Binario – esadecimale

(100 1100 0101) $(4\ C\ 5)_{16}$
4 C 5

ogni gruppo di 4 bit corrisponde ad una cifra esadecimale

Esempio 2

Addizione binaria

Una volta definita la rappresentazione dei numeri nel sistema binario, vediamo le operazioni che si possono compiere su di essi. La più semplice operazione che si può compiere, qualunque sia la base utilizzata, è l'addizione. Come nel caso decimale, se si conosce il risultato dell'addizione di 2 cifre qualsiasi, è possibile, operando 'per colonne', sommare numeri di qualunque lunghezza.
Nel caso decimale le 10 cifre da 0 a 9 danno origine a 100 (10^2) combinazioni possibili.

Nel caso binario, se si considera la somma di due bit, si possono verificare solamente quattro (2^2) possibili combinazioni:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10 (rappresentazione binaria di 2; in questo caso, eseguendo l'addizione, il risultato è zero e si riporta 1 sulla colonna alla sinistra di quella su cui si sta operando, così come accade quando si effettua la somma 9+1 in decimale).

Per fare un'addizione si opera come nel caso di un'addizione decimale: l'Esempio 3 mostra come eseguire la somma fra interi positivi 152 + 86, avendo a disposizione 8 bit per la loro rappresentazione, quindi potendo rappresentare i 256 (2^8) numeri da 0 a 255.

ADDIZIONE BINARIA

Se si opera su singoli bit

0 + 0 = 0
0 + 1 = 1 **N.B. $(10)_2 = (2)_{10}$**
1 + 0 = 1
1 + 1 = 10 (cioè, facendo l'addizione, scrivo zero e riporto 1)

In decimale:
(1)
1 5 2 +
8 6 **(N.B. 5 + 8 = 13 scrivo 3 e riporto 1)**
---
2 3 8

In binario:
$(152)_{10} = (10011000)_2$
$(86)_{10} = (01010110)_2$

Applicando le 4 'regole' alle coppie di bit corrispondenti

(1)
10011000 +
01010110

11101110

Esempio 3

Quando tuttavia si lavora al limite delle capacità di rappresentazione del calcolatore esistono opportuni circuiti e opportuni bit che evidenziano condizioni di errore. Ad esempio possono verificarsi situazioni in cui il risultato di un'operazione è troppo grande per essere rappresentato in modo corretto, come accade se si cerca di fare la somma 1+1 utilizzando una rappresentazione su un solo bit. In tal caso sarebbero necessari 2 bit per rappresentare correttamente il risultato $(10)_2$.
Questa condizione, in cui il numero di cifre necessarie a rappresentare in modo esatto un numero è maggiore delle cifre effettivamente disponibili per la sua rappresentazione, si dice *condizione di overflow (traboccamento)*.
Se, ad esempio, anziché 86 si fosse voluto sommare a 152 il numero 118 (v. Esempio 4), il risultato sarebbe stato 270. Ma 270 è maggiore di 255, cioè maggiore del più grande numero rappresentabile con 8 bit. In tal caso, in corrispondenza della cifra più significativa dei due addendi si sarebbe ottenuto come risultato 0 con riporto 1. Tuttavia non ci sarebbe stato spazio per rappresentare il bit derivante dal riporto negli 8 bit a disposizione per il risultato.
All'interno del calcolatore questo bit 'in eccesso' viene inserito in un apposito registro della CPU (v. seguito) e si chiama *bit di overflow*: esso indica che si sono superate le capacità di rappresentazione della macchina.

CONDIZIONE DI OVERFLOW

152 + 10011000+
118 01110110

270 (1) 00001110 00001110

270 non è rappresentabile con 8 bit!

bit di overflow (riporto derivante dalla somma dei due bit più significativi)

Esempio 4

L'operazione di addizione fra numeri in base 2 viene eseguita quindi in modo del tutto analogo a quella fra numeri in rappresentazione decimale. Del resto, come detto, entrambe le notazioni sono di tipo posizionale e sono quindi soggette alle stesse regole.

Sottrazione binaria (rappresentazione di numeri relativi)

Anche per la sottrazione si potrebbe operare in modo del tutto simile al caso decimale. Tuttavia, utilizzando una opportuna rappresentazione, è possibile fare in modo che sottrazioni e addizioni possano essere effettuate utilizzando la stessa operazione e quindi, all'interno del calcolatore, utilizzando gli stessi circuiti.
Sinora si sono considerati solo numeri *interi positivi* (detti anche *numeri naturali*). Per poter fare le sottrazioni si introducono i *numeri relativi*, che permettono di rappresentare quantità numeriche sia positive che negative. Nell'algebra classica, a livello notazionale, si rendono uniformi i concetti di addizione e sottrazione facendo equivalere l'operazione a - b alla somma a + (-b). Perché questa equivalenza si riscontri anche quando si devono eseguire, in modo meccanico, addizioni fra numeri relativi (quindi addizioni e sottrazioni secondo la stessa regola meccanica) e perché tali operazioni possano essere realizzate secondo le stesse regole viste per i numeri naturali, è necessario ricorrere alla cosiddetta *aritmetica del complemento*.

Se utilizziamo la rappresentazione binaria, si definisce *complemento a 2* di un numero m, rappresentato con N bit, la differenza fra 2 elevato alla N ed m, cioè

$C(m) = 2^N - m$

In generale, data una base b e una rappresentazione su N cifre in base b si definisce *complemento a b* di m il numero $C(m) = b^N - m$

Si noti che, qualunque sia la base, l'operazione di complemento stabilisce una corrispondenza biunivoca fra un numero e il suo il complemento. Infatti, il complemento del complemento di un numero è il numero stesso: $C(m) = b^N - m$; quindi: $C(C(m)) = C(b^N - m) = b^N - (b^N - m) = b^N - b^N + m = m$.

Nell'aritmetica del complemento, i numeri negativi (e solo quelli!) si rappresentano mediante quella che, se si stesse rappresentando un numero naturale, corrisponderebbe alla rappresentazione del complemento del loro valore assoluto. Ad esempio, la rappresentazione in complemento a 2 con N = 8 bit del numero -32 corrisponde alla rappresentazione in base 2 del numero naturale $2^8 - 32 = 256 - 32 = 224$, cioè del complemento a 2 di 32 (v. Esempio 5a)

COMPLEMENTO DI UN NUMERO

Data una rappresentazione su N bit, si definisce COMPLEMENTO A 2 DI UN NUMERO m il numero $2^N - m$, e si indica con C(m)

$C(m) = 2^N - m$

(a)

$(32)_{10} = (00100000)_2$ (N = 8)

$C(32) = 2^8 - 32 = 256 - 32 = 224 = (11100000)_2$

(b)

Si noti che $C(m) = 2^N - m = (2^N - 1) - m + 1$
Quindi $C(32) = (255 - 32) + 1$

Se calcoliamo 255 - 32 (= 223), il risultato espresso in binario è

11011111

Ma 32 è uguale a 00100000 (cioè lo stesso numero con 0 e 1 scambiati)

Quindi, per rappresentare $[(2^N - 1) - m]$ basta trasformare in 1 ogni 0 di m e in 0 ogni 1

Aggiungendo poi 1 (poiché $C(m) = (2^N - m)$)

```
11011111 +
       1
--------
11100000
```

ottengo lo stesso risultato dell'esempio (a).

Quindi, come regola meccanica, si può dire che per ottenere il complemento di un numero si trasformano in 0 gli 1 e viceversa, aggiungendo infine 1 al risultato ottenuto.

Esempio 5

Esiste, comunque, una regola meccanica che semplifica l'operazione di complementazione (v. Esempio 5b). Esprimiamo il complemento a 2 di un numero come:
$C(m) = 2^N - m = [(2^N - 1) – m] + 1$
Si osserva che, nel caso N=8, $2^N - 1$ è pari a 255, ovvero, in binario, a 11111111. Analogamente, qualunque sia N, il numero naturale $2^N – 1$ è sempre rappresentato da una stringa di N bit, tutti uguali a 1. Se si sottrae un qualunque numero m da tale valore è facile vedere che il risultato è un numero rappresentato come il numero da complementare, in cui tuttavia ogni 0 viene sostituito da un 1 e viceversa. Abbiamo quindi ottenuto una semplice regola meccanica che, in un solo passo, permette di calcolare la quantità $[(2^N - 1) – m]$. Per ottenere il complemento a 2 di m non resta quindi che sommare 1 al risultato così ottenuto.
Si noti che, nella rappresentazione degli interi in complemento a 2, i numeri che hanno 0 come MSB sono positivi e possono essere interpretati come se fossero numeri naturali, mantenendo quindi consistenza con la rappresentazione utilizzata per questi ultimi. Al contrario, quelli che hanno 1 come MSB rappresentano numeri negativi (v. Esempio 6) e devono essere decodificati con la regola del complemento, tenendo cioè conto che rappresentano numeri negativi il cui valore assoluto (il numero privato del segno) è $2^N – m$, dove m è il valore che rappresenterebbe la sequenza di bit che stiamo considerando se la interpretassimo come numero naturale.

SOTTRAZIONE BINARIA

46 **46 è positivo, quindi segue la normale rappresentazione**
-127 **-127 va invece rappresentato con il complemento**

- 81

$(46)_{10} = (00101110)_2$
$(-127)_{10} = C(127) = (10000001)_2$

Infatti $(127)_{10} = (01111111)_2$
Usando il semplice procedimento visto in precedenza si scambiano 0 e 1, ottenendo 10000000 (equivale a fare 255 - 127); poi si aggiunge 1, e si ottiene 10000001

46 – 127 = 46 + (-127) .

Eseguo l'addizione:

46 +	**00101110 +**
- 127 =	**10000001 =**
- 81	**10101111**

VERIFICA: Il risultato è un numero che inizia per 1, quindi è la rappresentazione complementata di un numero negativo. Per conoscerne il valore assoluto bisogna operare la complementazione.

C (10101111) = 01010000 + 1 = 01010001 = $(81)_{10}$,

Quindi il risultato è - 81

Esempio 6

Pertanto, utilizzare la rappresentazione in complemento a 2 consente di usare lo stesso procedimento (e lo stesso circuito all'interno del calcolatore) per effettuare sia le addizioni che le sottrazioni. L'unica differenza rispetto all'addizione fra numeri naturali riguarda la condizione di overflow. Nel caso di addizioni fra numeri naturali, infatti, ogni qualvolta il bit di overflow (che coincide col riporto derivante dalla somma dei bit più

significativi) vale 1, ciò indica che si è superata la capacità della parola utilizzata per la rappresentazione; si genera quindi una condizione di errore. Nel caso invece dell'estensione dell'addizione ai numeri negativi non basta che il bit di overflow sia 1, ma è necessario confrontare il bit di overflow con il riporto che si è avuto sulla cifra più significativa; se questi due bit sono diversi, si ha una condizione di errore; se sono uguali, il risultato è corretto (v. Esempio 7) .
Trascurare la condizione di overflow può portare a risultati abbastanza curiosi. Ad esempio, utilizzando 8 bit per rappresentare i numeri relativi, se consideriamo il numero 127 e sommiamo 1, otteniamo una rappresentazione che, interpretata come numero naturale, equivarrebbe a 128; con la convenzione vista per i numeri relativi, invece, essa è la rappresentazione di -128. Quindi, è importante avere sempre presente, una volta che sia siano stabiliti tipo e lunghezza (numero di bit) della rappresentazione, l'intervallo entro il quale i numeri devono essere compresi per essere rappresentati in modo corretto. Se si rappresentano *numeri naturali*, con le 2^N combinazioni ottenibili con N bit si possono rappresentare i 2^N numeri **da 0 a 2^N -1**. Utilizzando invece la rappresentazione complementata per rappresentare i *numeri relativi*, i 2^N numeri rappresentabili con N bit dovranno essere divisi in ugual misura fra numeri negativi e numeri positivi (che comprendono anche lo zero); quindi con N bit si potranno rappresentare i numeri che vanno **da -2^{N-1} a $+2^{N-1}$ –1,** cioè ad es. da –128 a 127, se si usa una rappresentazione su 8 bit.

CONDIZIONE DI OVERFLOW NELLA RAPPRESENTAZIONE COMPLEMENTATA

Se si usa l'aritmetica del complemento, non basta osservare il solo bit di overflow, ma si deve confrontare il bit di overflow con il riporto avuto sull'ultima cifra (bit di *carry*).

Se i 2 bit sono *diversi* **si ha CONDIZIONE DI OVERFLOW**

Se i 2 bit sono *uguali* **non si ha CONDIZIONE DI OVERFLOW.**

Se esaminiamo l'esempio 6:

		(0) < ---------- bit di carry
46 +		00101110 +
-127 =		10000001
-81	**(0)***	10101111

*** = bit di overflow (riporto dovuto alla somma delle ultime due cifre)**

I 2 bit sono uguali e quindi il risultato, rappresentato con gli 8 bit a disposizione, è corretto.

Se invece avessimo provato a fare:

		(0)
-20 +		11101100 +
-110		10010010
-130	(1)	01111110

Questa volta i 2 bit sono diversi. Infatti il risultato rappresentato dagli 8 bit, corrispondente a +126, non è quello desiderato. Infatti con 8 bit si possono rappresentare numeri compresi nell'intervallo da -128 a +127.

Esempio 7

Riepilogando, è stato dapprima mostrato come rappresentare e sommare numeri interi positivi (naturali); poi, introducendo la notazione complementata, si è mostrato come sia possibile utilizzare la stessa operazione per effettuare l'addizione fra interi relativi (che assumono valori positivi o negativi). La sottrazione di un numero diviene quindi equivalente alla somma col corrispondente numero negativo e può essere effettuata seguendo le stesse regole utilizzate per sommare i numeri naturali.

Diamo infine un rapido sguardo anche alle divisioni ed alle moltiplicazioni; esiste una proprietà della notazione posizionale per la quale, se si fa scorrere la rappresentazione di un numero (es. il numero decimale 105) verso sinistra (ottenendo 1050), si moltiplica di fatto quel numero per la base; infatti in questo caso 1050 = 105*10. Analogamente, dato un numero binario, ad es. 1101, che è la rappresentazione binaria di $(13)_{10}$, se si compie la stessa operazione, cioè uno *shift* (scorrimento) verso sinistra, si ottiene 11010, che corrisponde al numero decimale 26. Anche in questo caso, uno shift di una posizione verso sinistra ha come conseguenza la moltiplicazione del numero di partenza per la base, in questo caso per 2. Analogamente si può vedere che lo scorrimento di un numero di una posizione verso destra equivale alla divisione del numero stesso per la base. Nel caso del numero decimale 105, considerando solo la parte intera del quoziente (stiamo comunque operando su numeri interi), si ottiene 10, che è il quoziente di 105:10. Anche nel caso binario, shiftando verso destra 1101 (rappresentazione di 13), si ottiene 110, che è la rappresentazione binaria di 6, che è il quoziente della divisione fra interi 13:2.

RAPPRESENTAZIONE DI NUMERI INFERIORI ALL'UNITA'

Vediamo ora come si possono rappresentare in base 2 i numeri inferiori all'unità (v. Esempio 8). Partiamo da un esempio in base 10: il numero 19.375 significa, per ciò che riguarda la parte intera, $1x10^1 + 9x10^0$. Continuando dopo la virgola, la regola applicata resta la stessa: infatti, proseguendo verso destra, si continua a moltiplicare le cifre corrispondenti per potenze decrescenti di 10, quindi .375 rappresenta $3x10^{-1} + 7x10^{-2} + 5x10^{-3}$, cioè 3/10 + 7/100 + 5/1000.
Anche a livello di numerazione binaria possiamo compiere esattamente la stessa operazione.

RAPPRESENTAZIONE (IN VIRGOLA FISSA) DEI NUMERI INFERIORI ALL'UNITA'

Base 10

19.375 = **1**$x10^1$ + **9**$x10^0$ + **3**$x10^{-1}$ + **7**$x10^{-2}$ + **5**$x10^{-3}$

Base 2

10011.011 = **1**$x2^4$ + **0**$x2^3$ + **0**$x2^2$+ **1**$x2^1$ + **1**$x2^0$ + **0**$x2^{-1}$ + **1**$x2^{-2}$ + **1**$x2^{-3}$

CONVERSIONE PARTE INTERA	**CONVERSIONE PARTE DECIMALE**
19:2 = 9 (resto **1**)	
9:2 = 4 (resto **1**)	0.375 x 2 = **0**.75
4:2 = 2 (resto **0**)	0.75 x 2 = **1**.5
2:2 = 1 (resto **0**)	0.5 x 2 = **1**.0 => **011**
1:2 = 0 (resto **1**) => **10011**	

Esempio 8

Vediamo come convertire un numero inferiore all'unità nella sua rappresentazione binaria. Si moltiplica per 2 il numero da trasformare e si annota la parte intera del risultato. Si considera poi la parte inferiore all'unità del risultato e la si moltiplica nuovamente per 2, ripetendo l'operazione finché la parte inferiore all'unità non diventa zero o fino a quando la successione dei risultati non inizia a ripetersi (numero periodico). La

rappresentazione binaria della parte inferiore all'unità è quindi data dalla sequenza delle parti intere dei risultati ottenuti, nel nostro caso 011.
Non tutti i numeri che hanno una rappresentazione finita in una base, cioè per rappresentare i quali in tale base è sufficiente un numero finito di cifre, hanno una rappresentazione finita anche in un'altra base. Si pensi al caso di 1/3 che, in base 3, può essere rappresentato come 0.1, mentre in base 10 è un numero periodico 1/3 = 0.3333333……
L'unico caso in cui si ottiene sempre un numero finito di cifre è la conversione dalla base di partenza ad una base che è una sua potenza. Se è possibile ottenere una rappresentazione finita in base 2 per un numero, avremo una rappresentazione finita per esso anche in base 4, 8, 16 e così via, come si è visto nel caso della conversione di un numero da binario a esadecimale; si è visto infatti che essa consiste nella conversione di successivi raggruppamenti di 4 cifre binarie nelle corrispondenti cifre esadecimali.
Si consideri (Esempio 9) il numero 0.31, che ha una rappresentazione finita in base 10. Operiamo la conversione in binario usando lo stesso procedimento visto in precedenza.
Si ottiene un numero periodico che approssima, ma non rappresenta esattamente, il numero di partenza.

$(0.31)_{10}$

0.31 x 2 = 0.62
0.62 x 2 = 1.24
0.24 x 2 = 0.48
0.48 x 2 = 0.96
0.96 x 2 = 1.92
0.92 x 2 = 1.84
0.84 x 2 = 1.68
0.68 x 2 = 1.36
0.36 x 2 = 0.72
0.72 x 2 = 0.44
0.44 x 2 = 0.88
0.88 x 2 = 1.76
0.76 x 2 = 0.52
0.52 x 2 = 1.04
0.04 x 2 = 0.08
0.08 x 2 = 0.16
0.16 x 2 = 0.32
0.32 x 2 = 0.64
0.64 x 2 = 1.28
0.28 x 2 = 0.56
0.56 x 2 = 1.12
0.12 x 2 = 0.24 ……

$0.0100111110101010000101\overline{}$

Esempio 9

Questa proprietà crea problemi nel caso in cui sia necessario ottenere la stessa precisione che si avrebbe lavorando in base 10. Ad esempio, nelle applicazioni gestionali, se si eseguono molte operazioni in sequenza, anche piccoli arrotondamenti possono portare ad errori rilevanti. In questi casi si può usare la cosiddetta rappresentazione BCD (Binary Coded Decimal): si rappresentano cioè i numeri decimali cifra per cifra. I dieci simboli da 0 a 9, che sono utilizzati in base 10, si possono rappresentare ciascuno coi corrispondenti gruppi di 4 bit. Utilizzando poi meccanismi diversi per eseguire le operazioni, ogni cifra decimale viene rappresentata singolarmente da un gruppo di 4 bit. Si ottengono quindi sempre rappresentazioni finite e la stessa precisione che si avrebbe operando in decimale. Ci sono però degli inconvenienti: con 4 bit si potrebbero rappresentare 16 valori; in questo caso invece si utilizzano 4 bit per rappresentare 10 valori; quindi, con 4 cifre BCD, cioè con 16 bit, si possono rappresentare i numeri da 0 a 9999, cioè 10000 numeri, mentre la rappresentazione binaria consentirebbe, con 16 bit, di rappresentare 65536 numeri diversi numeri, ad es. i naturali da 0 a 65535. Ciò comporta che la rappresentazione BCD sia

ancora meno compatta di quella binaria; poiché inoltre, per effettuare le operazioni, si devono utilizzare regole diverse e più complesse da quelle che sono normalmente utilizzate dal calcolatore quando esegue operazioni direttamente in binario, queste vengono eseguite dal calcolatore ad una velocità notevolmente inferiore.
Normalmente, un programma che richiede una maggiore occupazione di memoria è più veloce di un programma che esegue le stesse operazioni utilizzando meno memoria e viceversa. In questo caso, sotto entrambi i punti di vista, si ha un peggioramento delle prestazioni del calcolatore: si usa più memoria e si eseguono più lentamente le operazioni, pur di avere una rappresentazione esatta dei numeri. Questa notazione resta quindi confinata ad applicazioni particolari; ad esempio, per applicazioni gestionali sono stati definiti linguaggi, come ad es. il COBOL, che possono utilizzare questa notazione per ottenere una maggiore precisione. Si possono rappresentare anche numeri con parte inferiore all'unità sottintendendo che la virgola si trovi in una posizione prefissata e continuando a rappresentare ogni cifra decimale con la rappresentazione BCD.

BCD (BINARY CODED DECIMAL)

I numeri decimali si rappresentano, cifra per cifra, a gruppi di 4 bit.
Es.: $(1579)_{10} = (0001010101111001)_{BCD}$

Infatti le sequenze di 4 bit
0001 0101 0111 1001
rappresentano le 4 cifre decimali
1 5 7 9

Esempio 10

Rappresentazione in virgola mobile

Abbiamo sinora parlato della rappresentazione binaria dei numeri inferiori all'unità senza porre alcun limite alla lunghezza delle parole utilizzate. Nel calcolatore si usano parole di lunghezza finita, ad esempio 8, 16 o 32 bit. E' quindi necessaria una rappresentazione che permetta di rappresentare qualunque numero decimale con la massima precisione consentita dalla lunghezza della rappresentazione utilizzata.
A tal fine si utilizza la cosiddetta *rappresentazione in virgola mobile* (*floating point*). E' una notazione di tipo esponenziale, in cui la sequenza di N bit utilizzata per rappresentare un numero reale viene suddivisa in 3 parti: un bit di segno, un certo numero di bit che rappresenta la cosiddetta *caratteristica* (o *esponente*) e, infine, la cosiddetta *mantissa*, rappresentata di solito con un numero di bit superiore a quello dell'esponente. Quindi, la rappresentazione di un numero in virgola mobile è una sequenza binaria ottenuta dalla concatenazione delle rappresentazioni di 3 "numeri" diversi, aventi ognuno una propria codifica e un proprio significato. Il primo è il *segno*, costituito da un unico bit (0 per i numeri positivi e 1 per i numeri negativi). La mantissa e l'esponente, invece, rappresentano il valore assoluto del numero in notazione esponenziale, che rappresenta un generico numero n nel formato $n = +/- m \times B^e$ (m= mantissa, B= base, e= esponente). Utilizzando una rappresentazione di questo tipo, si potrebbe rappresentare uno stesso numero in infiniti modi diversi. Ad es., in base 10, 100 può essere espresso come 100×10^0, o 1×10^2, o anche come 0.1×10^3, ecc., come mostrato nell'Esempio 11.
Tali rappresentazioni del numero 100 corrispondono, rispettivamente, a mantissa=100 ed esponente=0, a mantissa=1 ed esponente=2, e a mantissa=0.1 ed esponente=3. Per rendere univoca la rappresentazione di un numero in virgola mobile si impone che la mantissa sia sempre inferiore all'unità e abbia la cifra più significativa diversa da 0; si sottintende cioè che la virgola sia posizionata a sinistra della cifra più significativa della mantissa (*normalizzazione della mantissa*). Se lavoriamo in binario, B è fissato ed è ovviamente uguale a 2. Per quello che riguarda il segno, la regola è quella appena ricordata. Resta quindi da vedere come codificare la mantissa e l'esponente, dato il numero che si vuole rappresentare.

La mantissa è una sequenza di bit che comprende tutte le cifre significative del numero, cioè tutti i bit necessari a rappresentarlo in modo esatto, a partire dall'1 più significativo all'1 meno significativo, ammesso che la lunghezza della rappresentazione utilizzata lo consenta. Il segno, come già detto, è rappresentato separatamente mediate un bit ad esso "dedicato". La mantissa viene rappresentata quindi con la stessa convenzione utilizzata per la rappresentazione del valore assoluto di un numero in virgola fissa, supponendo che la virgola si trovi alla estrema sinistra del numero. La sequenza di bit che rappresenta la mantissa è quindi esattamente la stessa che si otterrebbe rappresentando il numero in virgola fissa e spostando la virgola all'immediata sinistra della cifra più significativa.

RAPPRESENTAZIONE IN VIRGOLA MOBILE

Stabilita la lunghezza della parola, essa viene suddivisa in 3 parti

Segno | Mantissa | Esponente

Si usa una notazione di tipo esponenziale; cioè si esprime un numero n come
$n = +/- m \times B^e$ (B = base)

In questo modo si possono avere più rappresentazioni per uno stesso numero.

Es. (in base 10, cioè B=10)

100	**1×10^2**	**m = 1**	**c = 2**
	100×10^0	**m = 100**	**c = 0**
	0.1×10^3	**m = .1**	**c = 3**

Esempio 11

Ad esempio, $(10.25)_{10}$ in virgola fissa corrisponde a $(1010.01)_2$. Pertanto la mantissa è in questo caso rappresentata dalla sequenza di bit 101001 che corrisponde, per la convenzione cui si è appena accennato, al numero $(0.101001)_2$. Se le cifre a disposizione per rappresentare la mantissa sono meno delle cifre significative necessarie a rappresentare in modo esatto il numero che dobbiamo convertire, si trascurano i bit meno significativi (quelli cioè che si trovano alla estrema destra del numero), fino a raggiungere il numero di bit richiesto. Ad esempio, se avessimo avuto a disposizione soltanto 5 bit per rappresentare la mantissa di $(10.25)_{10}$ questa sarebbe diventata 10100, trascurando il bit meno significativo.
Tuttavia, normalizzando la mantissa, nell'esempio fatto è stata operata una divisione del numero per 2^4 (si ricordi che uno scorrimento a sinistra della virgola di N posizioni, in qualunque base, corrisponde alla divisione del numero di partenza per la base elevata alla N). Per ottenere il valore che si vuole rappresentare è quindi necessario moltiplicare la mantissa per la stessa quantità per cui esso è stato diviso. Nell'esempio fatto, quindi, è necessario moltiplicare per 2^4. Questo determina automaticamente il valore dell'esponente che, nell'esempio, sarà quindi pari a 4, cioè $(100)_2$. L'esponente, tuttavia, può assumere valori anche negativi (si provi, ad esempio, a compiere le stesse operazioni per convertire il numero 0.25, che, con le convenzioni viste viene rappresentato come $+ .1 \times 2^{-1}$) e quindi la sequenza di bit che lo rappresenta deve essere in grado di rappresentare anche numeri negativi. Perciò, per codificare l'esponente si deve usare la rappresentazione in complemento a 2, poiché l'esponente è un numero relativo. Nel caso del numero decimale 0.25, utilizzando 4 bit per l'esponente, ad esempio, la sequenza di bit che codifica in binario l'esponente risulta 1111, cioè la codifica su 4 bit di –1.

Riepilogando, dato un numero n da convertire in virgola mobile con *k* bit di mantissa ed *e* bit di esponente:

- Il bit di segno è 0 se n è positivo; 1 se n è negativo
- Per determinare la mantissa si converte il valore assoluto del numero in virgola fissa: la mantissa è costituita dalla sequenza delle *k* cifre più significative di tale rappresentazione.

- Si sottintende che la rappresentazione della mantissa sia normalizzata, cioè che la virgola si trovi all'estrema sinistra del numero, cioè che la cifra più significativa risulti moltiplicata per 2^{-1}
- L'esponente è un numero il cui valore rappresenta il numero di posizioni di cui la virgola è stata spostata per operare la normalizzazione delle mantissa. E' positivo se la virgola è stata spostata a sinistra e negativo se la virgola è stata spostata a destra. E' rappresentato in complemento a 2: pertanto, se è negativo, per la sua rappresentazione bisogna complementare il suo valore assoluto.

Supponiamo di lavorare con 8 bit di mantissa e 4 bit di esponente e rappresentiamo il numero 1.5 in virgola mobile: sappiamo che è uguale a 1 + ½ cioè, in binario, 1.1 . Si è detto che, per convenzione, la mantissa deve comprendere le cifre più significative del numero (le cifre che compongono il numero a partire dal primo 1), cioè la mantissa deve essere normalizzata. In questo caso, quindi, la rappresentazione della mantissa sarà 11000000 che rappresenta il numero binario 0.11 . Questa sequenza binaria corrisponde alla rappresentazione da cui si era partiti, in cui la virgola è stata spostata a sinistra di una posizione (si è diviso per 2 il numero, passando da 1.1 a 0.11). Per rappresentare correttamente il numero da convertire, bisogna tornare a moltiplicare la mantissa per 2 (cioè per 2^1), quindi si deve porre l'esponente = 1, cioè:
$1.1 = 0.11 \times 2^1$.

Nel caso di 0.25, invece, si avrà:
- segno = 0 (il numero è positivo)
- mantissa = 10000000, corrispondente al numero 0.1 (la rappresentazione di $(0.25)_{10}$ in virgola fissa è $(0.01)_2$)
- esponente = 1111, corrispondente a –1 in rappresentazione in complemento a 2 su 4 bit, poiché per passare da 0.01 a 0.1 il numero da convertire è stato moltiplicato per 2, cioè la virgola è stata spostata a destra di 1 posizione.

Per quanto riguarda le addizioni, per poter effettuare una somma con questa rappresentazione bisogna che i due numeri abbiano uguale esponente: questo equivale, quando si sommano due numeri, ad es. 123 e 1240, in base 10, a fare in modo che tali numeri siano incolonnati correttamente. Questo fa sì che la loro somma mantenga lo stesso esponente dei due numeri e abbia come mantissa la somma delle mantisse. Quindi lo zero che compare in 1240 dovrà essere nella stessa posizione del 3 in 123, cioè dovrà essere moltiplicato per una potenza di 10 analoga a quella della cifra meno significativa dell'altro numero. Se si vogliono sommare 123 e 1240 è possibile quindi sommare $.123 \times 10^3$ con 1.240×10^3 operando solo sulle mantisse; non si può effettuare un'addizione, ad es., tra 123×10^0 e 124×10^1 , incolonnando 123 e 124. Quindi, anche nel calcolatore, la prima operazione che viene effettuata prima di compiere una somma in virgola mobile è proprio quella di uguagliare gli esponenti in modo di poter limitare l'operazione soltanto alla somma delle mantisse. Nel caso in esame si deve passare da $.124 \times 10^4$ alla rappresentazione 1.240×10^3 in modo che entrambi i numeri siano rappresentati con lo stesso esponente. Ovviamente, se si lavora con un numero limitato di cifre per la mantissa, questi spostamenti possono richiedere di troncare certi numeri e quindi ad errori nell'operazione. Tali errori vengono minimizzati eliminando le cifre meno significative, ma fanno sì che il risultato rappresentato dal calcolatore in virgola mobile possa non essere esattamente quello che si otterrebbe lavorando con precisione infinita. Di questo bisogna tenere conto perché gli errori che si commettono in una lunga serie di calcoli possono propagarsi, accumularsi e portare a risultati che, per quanto il procedimento seguito sia corretto, possono anche essere del tutto errati o, comunque, approssimati in modo tropo grossolano.

Come visto anche nel caso dei numeri interi, fissata una rappresentazione che utilizza un numero finito di bit, anche se due numeri sono entrambi rappresentabili correttamente, non è detto che la loro somma lo sia; una cosa apparentemente strana che però può capitare lavorando in virgola mobile è che la somma di due numeri non risulti rappresentabile in modo esatto ma che un numero più grande lo sia. Ad esempio, se si considerano 10 e 0.5, supponendo di utilizzare una rappresentazione con 4 bit di mantissa, si ha che la rappresentazione in base 2 di 10 è 1010, quella in base 2 di 0.5 è 0.1. Se si esegue la somma dei due numeri,

il risultato è 1010.1. Poiché, tuttavia, si hanno solo 4 bit a disposizione per rappresentare tale risultato e quindi risulta necessario eliminare una cifra dalla rappresentazione, si esclude quella che ha il minor peso, cioè il bit meno significativo. Quindi la mantissa è 1010, poiché non siamo in grado di rappresentare 10101. Quindi l'addizione 10 + 0.5, utilizzando 4 bit per la mantissa, produce ancora 10 come risultato!
Se fra il bit più significativo e quello meno significativo di un numero ci sono più cifre di quante possono essere rappresentare, le cifre meno significative vengono eliminate (v. Esempio 12).
Si verifica quindi un apparente paradosso: non possiamo rappresentare in modo esatto 10.5 ma può capitare che un numero più grande, ad es. 11, torni ad essere rappresentabile, poiché bastano 4 cifre per rappresentarlo in modo esatto.
Una situazione particolarmente critica che può portare ad errori rilevanti è l'effettuazione di somme fra due numeri di cui uno è molto più grande dell'altro.

In una rappresentazione con 4 bit di mantissa e 4 di esponente, si considerino i due numeri:

10 1010
0.5 0.1

Per rappresentare la loro somma (1010.1) si hanno a disposizione solo 4 bit, mentre ne occorrerebbero 5; l'esponente permette solo di spostare la virgola

al massimo rappresento

$$1010 = 0.1010 \times 2^4$$

S = 0
M = 1010
C = 0100

Quindi

10 + 0.5 = 10 !

Osservazione:
10.5 non è rappresentabile ma 11. ad es, lo è. Infatti $(11)_{10} = (1011)_2$: infatti bastano 4 bit.

Esempio 12

Per dare una misura della precisione con cui possono essere effettuati i calcoli su una certa macchina, si definisce *precisione di macchina* il più piccolo numero che, sommato ad 1, consente una rappresentazione esatta della somma. Cioè, per valutare la precisione di macchina di un calcolatore, si effettuano tante somme 1 + N rendendo sempre più piccolo il numero N (dividendolo iterativamente per 2) finché non si arriva al risultato che 1 + N = 1: a quel punto, l'ultimo numero N che ha prodotto un risultato diverso da 1 è il più piccolo numero su cui si può lavorare, rispetto all'unità. Più piccolo sarà questo numero, maggiore sarà la precisione ottenibile utilizzando quel calcolatore, cioè tanto più grande sarà il numero di bit con cui il calcolatore rappresenta la mantissa.

RAPPRESENTAZIONE DI CARATTERI ALFANUMERICI

Per concludere la rassegna dei dati che possono essere rappresentati all'interno di un calcolatore, vediamo come sono rappresentati all'interno di un calcolatore i caratteri alfanumerici. Per interagire più facilmente con il calcolatore bisogna usare un linguaggio che sia il più possibile simile al linguaggio che usiamo tutti i giorni, e che si esprima quindi secondo un alfabeto che conosciamo. La codifica che permette di rappresentare all'interno del calcolatore anche caratteri alfanumerici è la cosiddetta codifica ASCII (American Standard Code for Information Interchange). Normalmente la codifica ASCII è basata su parole di 7 bit oppure di 8 bit, nel qual caso essa è detta codifica ASCII estesa.
I primi 32 codici, da 0 a 31, rappresentano i cosiddetti caratteri di controllo. Ad es. il numero 10 rappresenta il "Line Feed". Quando si invia alla stampante questo carattere il foglio si muove di una posizione in verticale e si posiziona sulla riga successiva. Se si vuole ottenere un "a capo" completo c'è bisogno di una coppia di caratteri: uno è il "Line Feed" (codice decimale 10), l'altro è il codice 13 che si chiama "Carriage Return" (ritorno di carrello), che fa ritornare il carrello con la testina di stampa fino al margine sinistro della pagina. Dalla ultima posizione (all'estrema destra) dell'ultima riga precedente, quindi, la testina di stampa si posiziona nella prima posizione della riga successiva.
I simboli, con codici da 32 a 47 sono simboli di interpunzione, +, /, *, ecc. Con i codici da 48 al 57 sono rappresentate le dieci cifre decimali, da 0 a 9. In particolare si può notare che, nel codice ASCII, hanno tutte una rappresentazione uguale a 0011 per i 4 bit più significativi seguiti dalla loro effettiva rappresentazione binaria come 4 bit meno significativi. Coi numeri da 65 a 90 sono rappresentate le lettere maiuscole; coi numeri da 97 al 122 sono rappresentate le lettere minuscole.

Se si usa la rappresentazione ASCII a 7 bit, si hanno a disposizione 128 simboli. Questa è la rappresentazione classica che si usava per le telescriventi: in essa sono comprese le lettere dell'alfabeto inglese, le 10 cifre decimali, i caratteri di controllo per gestire la telescrivente e i segni di interpunzione. Nei calcolatori, i codici che vanno da 128 a 255 sono utilizzati per la cosiddetta codifica ASCII estesa, che però non rispetta alcuno standard e quindi dipende da calcolatore a calcolatore. Tali codici corrispondono ad una serie di simboli che comprendono le lettere accentate ed altre lettere appartenenti ad alfabeti diversi da quello inglese, oltre ad alcuni simboli grafici, utili ad esempio per tracciare riquadri senza bisogno di operare con terminali di tipo grafico.