Problemi connessi all'utilizzo di un numero di bit limitato
Abbiamo visto quali sono i vantaggi dell'utilizzo della rappresentazione in complemento alla base: corrispondenza biunivoca fra rappresentazione binaria e numero decimale, semplificazione e riduzione del numero di circuiti e controllo efficace dell'overflow error. I limiti diventano evidenti dovendo trattare numeri molto grandi o numeri non interi. Utilizzando **2 byte** (come usualmente si fa) per la rappresentazione in complemento a due l'intervallo di rappresentazione è $[-2^{15}, 2^{15}-1]$ ossia [-32768, 32767]. Dovendo elaborare quantità numeriche come i dati finanziari di un'azienda, o la distanza fra la terra e la luna, o il numero di abitanti di una nazione è chiaro il limite dovuto al numero limitato di bit che si utilizzano. Per superare il problema dell'intervallo di rappresentazione limitato si possono adottare vari criteri:

- aumentare il parallelismo, vale a dire il numero di bit utilizzati
- eseguire il calcolo in più passi
- adottare una diversa rappresentazione che estenda, a parità di bit, l'intervallo di rappresentazione

In particolare si potrebbe estendere il numero di bit su cui rappresentiamo il numero. Dato che gli elaboratori lavorano in base 2, passiamo da 16 bit a 32 bit che è la potenza di 2 immediatamente successiva a 16. L'intervallo di rappresentazione su 32 bit sarà:

$[-2^{31}, 2^{31}-1]$ e dunque [-2.147.483.648, +2.147.483.647]

L'aumento di bit utilizzati causa un costo: maggiore memoria utilizzata e circuiti di calcolo su 32 bit invece che su 16 bit con il conseguente aumento del prezzo del calcolatore (tutte le macchine hanno ormai un'architettura a 32 bit).
In realtà vedremo com'è possibile superare sia il problema dell'intervallo di rappresentazione che quello dei numeri frazionari utilizzando una rappresentazione diversa da quella in complemento a 2.

Rappresentazione in virgola fissa (fixed point)
L'uomo distingue tramite un virgola o un punto (dipendentemente dal tipo di convenzione adottata, qui si utilizzerà il punto) la parte intera dalla parte frazionaria di un numero. Il sistema è sempre di tipo posizionale perché le cifre frazionarie hanno dei pesi che sono associati alle potenze della base ma con esponente negativo. In particolare l'esponente della cifra più a sinistra della virgola vale –1.

Il valore vero del numero (in base r) rappresentato dalle cifre $d_{n-1} \ldots d_2\ d_1\ d_0\ ,\ d_{-1}\ d_{-2} \ldots d_{-m}$ viene calcolato nel modo seguente:

$$N_v = d_{n-1} \times r^{n-1} + d_{n-2} \times r^{n-2} + \ldots + d_2 \times r^2 + d_1 \times r^1 + d_0 \times r^0 + d_{-1} \times r^{-1} + d_{-2} \times r^{-2} + \ldots + d_{-m} \times r^{-m}$$

Esempi:

$15.05_{(10)} = 1 \times 10^1 + 5 \times 10^0 + 0 \times 10^{-1} + 5 \times 10^{-2}$

$011.01_{(2)} = 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 2 + 1 + ¼ = 2 + 1 + 0.25 = 3.25$

$103.12_{(4)} = 1 \times 4^2 + 0 \times 4^1 + 3 \times 4^0 + 1 \times 4^{-1} + 2 \times 4^{-2} = 16 + 3 + 0.25 + 0.0625 = 19.3125$

In realtà all'interno dell'elaboratore non c'è la virgola ma una sequenza continua di cifre. Questo può portare a qualche problema:

supponiamo di avere i seguenti 4 bit: 0110

Se immaginiamo che la virgola sia posta dopo l'ultima cifra (ossia 0110.0) allora il numero vale +6; ma possiamo immaginare di posizionare la virgola in un punto qualsiasi dei 4 bit, ottenendo:

0110. valore: +6
011.0 valore: +3.0
01.10 valore: +1.0 + ½ = +1.5
0.110 valore: + ½ + ¼ = +0.75

Il che implica che nel calcolatore bisogna stabilire una *convenzione* che assuma la virgola in una posizione ben precisa. Per tale motivo si parla di notazione in *virgola fissa* (*fixed point*): la posizione della virgola viene stabilita e mantenuta fissata durante l'esecuzione dei calcoli.

Esempio:

Supponiamo d'avere due numeri:

0110 e 0100

Per 0110 supponiamo che la virgola sia fra il bit di posto 1 e quello di posto 2:
0110 (che equivale a 01.10) vale 1.5

Per 0100 supponiamo che la virgola sia fra il bit di posto 0 e quello di posto -1:
0100 (che equivale a 0100.) vale 4

le due sequenze così come sono non potrebbero essere sommate direttamente perché non allineeremmo i pesi corrispondenti e quindi dovremmo spostare le cifre a destra e/o a sinistra per effettuare la somma:

```
   01.10 +
0100.    =
0101.10   che vale 5.5
```

Per effettuare il calcolo senza prima allineare i due numeri avremmo dovuto stabilire a priori la posizione della virgola. Supponiamo di utilizzare 6 bit e di stabilire che la virgola è fissata fra il bit di posto 1 e quello di posto 2:

```
0001.10 +    ossia 1.5
0100.00 =    ossia 4
0101.10      ossia 5.5
```

Si può, quindi, affermare che l'aver fissato la posizione della virgola consente di effettuare i calcoli senza commettere errori. Normalmente si assume la posizione della virgola o nell'*estremo sinistro* (e quindi il numero è frazionario) o nell'*estremo destro* (e quindi il numero sarà intero).

Rimane il problema di stabilire univocamente la posizione della virgola e di trattare numeri molto grandi e molto piccoli.

Rappresentazione in virgola mobile (floating point)

La massa dell'elettrone è
0.00000000000000000000000000000091 Kg

La massa della terra è
597360000000000000000000 kg

Come possiamo gestire numeri di questo tipo e stabilire univocamente la convenzione sulla posizione della virgola?

Si pensi cosa sarebbe necessario fare se si volesse rappresentare grandezze di questo tipo all'interno di un elaboratore poiché il numero di cifre in base 2 è notevolmente maggiore di quello del numero in base 10. Anche solo scrivere su un foglio di carta questi numeri è difficoltoso!. L'alternativa è di scrivere il numero indicandone le cifre essenziali, ad esempio: 9.1×10^{-31} o, nel secondo caso, 5.9736×10^{24}. Questo tipo di scrittura viene denominato *notazione scientifica*. Spesso il 10 viene sostituito dal simbolo "e" ottenendo 9.1e-31 kg e 5.9736e24 kg rispettivamente.
Questa notazione all'interno degli elaboratori assume la terminologia di *notazione in virgola mobile* (*floating point*).

Un numero è esprimibile come:

$$N = \pm M \times r^E$$

dove:
± è il segno del numero (positivo o negativo)
M è la *mantissa* e rappresenta le cifre essenziali (è un numero in valore assoluto)
E è l'*esponente* cui elevare la base *r* (è un numero relativo)

Dato un qualsivoglia numero, si può esprimere secondo questa notazione. Ad esempio:

N = 1260 potrebbe esprimersi per esempio nei seguenti diversi modi:

1260×10^0
$1.260 \times 10^{+3}$
$0.126 \times 10^{+4}$
12600×10^{-1}

È evidentemente preferibile avere una rappresentazione unica come accade per la virgola fissa, e quindi bisogna cercare di definire uno *standard* di rappresentazione. A tal fine si utilizza la notazione scientifica in *forma normalizzata*:

la mantissa è espressa come numero frazionario compreso fra la base elevata a -1 e 1:

$r^{-1} \le M < 1$ (in base 10 avremo: $0.1 \le M < 1$)

In tal modo non può aversi ambiguità: si stabilisce una corrispondenza biunivoca fra entità numerica e sua rappresentazione. *Mantissa normalizzata* equivale ad affermare che la prima cifra dopo la virgola è diversa da zero.

Ad esempio:

0.002 non è in forma normalizzata
0.3 è in forma normalizzata

Supponiamo che la mantissa sia codificata su 3 cifre e l'esponente sia codificato su 2 cifre. Vogliamo calcolare l'intervallo di rappresentazione (il numero massimo ed il numero minimo esprimibile in queste condizioni). Il valore massimo lo otterremo quando mantissa ed esponente hanno valore massimo. Nel nostro caso $M = 0.999$ che, per semplicità, approssimiamo a 1 ed $E = +99$ e quindi $N_{MAX} = +0.999 * 10^{+99}$ che per l'approssimazione fatta è circa uguale a 10^{+99}. Analogamente il più piccolo valore è circa -10^{+99}, mentre il valore frazionario più piccolo è 10^{-99}.
Avendo accertato che questa rappresentazione dei numeri è adeguata a rappresentare gra ndi intervalli dobbiamo capire come trasferire il tutto alla base 2 per poterla utilizzare mediante l'elaboratore.
A tal scopo dobbiamo riscrivere la formula precedentemente vista secondo questa nuova notazione:

$$N = (-1)^S \times M \times r^E$$

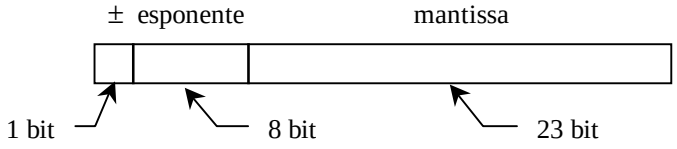
dove S vale o zero o uno

$-3.021_{(10)}$ verrebbe espresso con $(-1)^1 \times 0.3021 \times 10^{+1}$
$+3.021_{(10)}$ verrebbe espresso con $(-1)^0 \times 0.3021 \times 10^{+1}$

Avendo fissato la base r dobbiamo memorizzare solo le 3 entità (S; M; E). Nel caso del numero -3.021 (1; 0.3021; +1)

Codifica della virgola mobile nell'elaboratore
La base è $r = 2$. La mantissa M è, nella forma normalizzata, un numero frazionario compreso fra 2^{-1} e 1, ossia $0.5 \leq M < 1$ (M in pratica viene rappresentato in virgola fissa con la virgola posta all'estrema sinistra della sequenza di bit). L'esponente è un numero relativo che viene rappresentato in complemento a 2 (in realtà si usa la *rappresentazione con l'esponente polarizzato*).
Ricordandoci che l'elaboratore lavora su un insieme di bit che è potenza di 2, nella maggior parte dei calcolatori la rappresentazione dei numeri in virgola mobile avviene su **32 bit** così suddivisi:

il primo bit per il segno (quello a sinistra: 0 = segno positivo, 1 = segno negativo)
8 bit per l'esponente
23 bit per la mantissa.



Vediamo l'intervallo di rappresentazione.
Ricordandoci la formula scritta precedentemente, il massimo numero positivi sarà dato da 0.111111...1 ossia circa 1 che moltiplichiamo per 2 elevato al massimo valore intero codificabile su 8 bit in complemento a due, che è +127, quindi:

$N_{MAX} = +0.111111\ldots1 \times 2^{+127} \approx +10^{38}$
$N_{MIN} = -10^{38}$

Facendo un confronto dei valori ottenuti a parità nella notazione in virgola fissa utilizzando il complemento a due ($[-2^{31}, 2^{31}-1] = [-2.147.483.648, +2.147.483.647]$), si comprende subito il grosso vantaggio del sistema di rappresentazione in virgola mobile. Inoltre *si possono rappresentare numeri frazionari* molto piccoli, ad esempio: $0.000000\ldots01 \times 2^{-128} = 0.1368\text{e-}47 \approx 10^{-48}$

È possibile utilizzare una notazione ancora più spinta del floating point, la *doppia precisione* (*double*). Essa consiste di **64 bit** (8 byte):

1 bit per il segno
11 bit per l'esponente
52 bit per la mantissa.

Tale notazione estende ulteriormente l'intervallo di rappresentazione.

Perché non si usa sempre la notazione in virgola mobile:

- nell'intervallo di rappresentazione dei numeri, per esempio $[-10^{38}, +10^{38}]$ la distribuzione dei numeri rappresentabili non è uniforme
- complessità dei calcoli (per fare la somma fra due numeri espressi in virgola mobile è richiesto più tempo e più circuiti che non 2 numeri rappresentati in virgola fissa)

Distribuzione non uniforme
Possiamo spiegare tale problema mediante un esempio. Supponiamo che la mantissa normalizzata sia esprimibile su 2 bit. I valori possibili sono:

.00 che vale 0
.01 che non è utilizzabile perché < 0.5 ($= 2^{-1}$)
.10 che rappresenta 0.5
.11 che vale 0.75

Il nostro numero può essere espresso come mantissa solo dai valori 0.5 e 0.75.

Attribuiamo all'esponente alcuni valori:
se $E = -2$ i numeri rappresentabili sono: $0.5 \times 2^{-2} = 0.5 \times 0.25 =$ **0.125** e $0.75 \times 0.25 =$ **0.1875**
se $E = -1$ i numeri rappresentabili sono: $0.5 \times 2^{-1} = 0.5 \times 0.5 =$ **0.25** e $0.75 \times 0.5 =$ **0.375**
se $E = 0$ i numeri rappresentabili sono: $0.5 \times 2^{0} = 0.5 \times 1 =$ **0.5** e $0.75 \times 1 =$ **0.75**
se $E = 1$ i numeri rappresentabili sono: $0.5 \times 2^{1} = 0.5 \times 2 =$ **1** e $0.75 \times 2 =$ **1.5**

Riassumendo, usando 2 bit per la mantissa e 2 per l'esponente si ottengono i seguenti valori: ±0.125, ±0.1875, ±0.25, ±0.375, ±0.5, ±0.75, ±1, ±1.5 (± nasce dal bit di segno).

Si osservi che nell'intervallo [0.125, 0.25) abbiamo 2 numeri; nell'intervallo [0.25, 0.5), che è il doppio di quello precedente abbiamo ancora 2 numeri; nell'intervallo [0.5, 1) che è ancora il doppio di quello precedente abbiamo la possibilità di rappresentare ancora 2 numeri; nell'intervallo [1, 2) abbiamo infine ancora 2 numeri. Man mano che ci muoviamo nella direzione crescente di valori in potenze di due (da 0.125 a 2) la *densità* dei numeri diminuisce. Abbiamo quindi una distribuzione non uniforme.
Questa considerazione ci fa pensare al concetto di *precisione*. Nell'intervallo [0.125, 0.25) possiamo rappresentare solo 0.125 e 0.1875, ma non possiamo rappresentare l'entità 0.1415 se non attraverso un errore in difetto (0.125) o in eccesso (0.1875). È evidente che non potremo rappresentare perfettamente tutti i numeri reali ma ci dovremo accontentare di un'approssimazione, tanto più efficiente quanti più bit destiniamo alla mantissa.

Complessità dei calcoli
Senza voler entrare in dettaglio basti osservare che supponendo che per la somma fra 2 numeri in virgola fissa ci vuole un tempo di 1 microsecondo (1 milionesimo di secondo), per la somma di 2 numeri in virgola mobile ci vogliono almeno 10 microsecondi (facendo una quantità molto elevata di operazioni la differenza in termini di tempo non è per nulla trascurabile).