UNIVERSITÀ
degli STUDI
di CATANIA

Fondamenti di Informatica

Corso di Laurea in Ingegneria Elettronica

Corso di Laurea in Ingegneria Informatica

Rappresentazione dell'informazione: Teoria

La presente dispensa è stata prodotta dal Prof.S. Cavalieri, ed è ufficialmente adottata dai corsi di Fondamenti di Informatica per Ingegneria Informatica/Elettronica.

Rappresentazione dell'informazione

1. Introduzione

L'architettura di un calcolatore impone delle regole di conversione tra le rappresentazioni "umane" delle informazioni e quella binaria, ossia basata sulla codifica binaria (cioè due soli simboli: 1 e 0), disponibile nel calcolatore.

Le informazioni che l'uomo gestisce sono essenzialmente di due tipi: numeriche e alfanumeriche. Le informazioni numeriche sono comunemente rappresentate dall'uomo utilizzando la base 10 (ossia le 10 cifre numeriche, da 0 a 9). Le informazioni alfanumeriche sono rappresentate dal set di caratteri dell'alfabeto (26 in tutto) più le 10 cifre numeriche (da 0 a 9) più altri simboli (ad esempio i simboli ', ?, !, +, -, /,ecc.).

Scopo di questo capitolo è fornire una descrizione di come le informazioni numeriche ed alfanumeriche vengano rappresentate in un calcolatore. Al fine di comprendere meglio il contenuto di questa dispensa, nel seguito di questa Introduzione verranno forniti alcuni concetti di base relativi ai sistemi numerici.

Un sistema numerico è determinato quando si fissano alcuni elementi che lo caratterizzano:

- Un insieme limitato di simboli (cifre) che rappresentano quantità intere prestabilite
- Le regole che devono essere applicate per costruire i numeri
    - Non Posizionali: il valore delle cifre è indipendente dalla loro posizione (es. M nella numerazione romana vale sempre 1000);
    - Posizionali: ad ogni posizione della cifra all'interno della rappresentazione è associato un peso.

Di particolare importanza sono i sistemi numerici di tipo posizionale e per questo motivo saranno gli unici che considereremo. Nei sistemi numerici posizionali il numero (non negativo) N è rappresentato come una sequenza di n simboli di:

$$N = d_{n-1}d_{n-2}.....d_2d_1d_0$$

Dove ogni simbolo d_i viene chiamato cifra.

Nei sistemi numerici posizionali, ogni cifra ha un peso diverso a seconda della posizione che occupa. Per esempio nel sistema numerico decimale, ad ogni cifra viene assegnato un peso proporzionale ad una potenza del 10. Più precisamente, alla cifra i-esima viene assegnato un peso pari a 10^i. Per convenzione la cifra i-esima di un codice numerico è la i-esima cifra partendo da destra e iniziando a contare da 0. Per esempio se consideriamo il numero decimale 10061974, la cifra di posizione 0 vale 4 e pesa 10^0. La cifra di posizione 1 vale 7 e pesa 10^1, la cifra di posizione

2 vale 9 e pesa 10^2. Generalizzando, la cifra di posizione i pesa 10^i. Il valore di un codice numerico si ottiene sommando i prodotti delle cifre per il rispettivo peso.

Riprendendo l'esempio precedente si ha infatti:

$$10061974 = 1 \times 10^7 + 0 \times 10^6 + 0 \times 10^5 + 6 \times 10^4 + 1 \times 10^3 + 9 \times 10^2 + 7 \times 10^1 + 4 \times 10^0$$

In generale dato un qualsiasi codice posizionale in base b ogni cifra può assumere un simbolo appartenente ad un alfabeto ordinato di b simboli D.

Dato il seguente codice posizionale in base b:

$$N_b = d_{n-1}d_{n-2}.....d_2d_1d_0$$

se ad ogni simbolo di D si assegna in modo univoco un numero intero compreso tra 0 e b-1 allora l'equivalente di N_b in decimale (che indicheremo con N_{10}) viene così calcolato:

$$N_{10} = \sum_{i=0}^{n-1} d_i \cdot b^i$$

2. Codici Numerici

La nascita dei calcolatori è stata principalmente determinata dalla necessità di disporre di strumenti in grado di eseguire rapidamente operazioni aritmetiche. La logica conseguenza è stata la formalizzazione della rappresentazione dei numeri, che nella nostra mente sono sempre espressi in forma decimale (utilizzando i dieci numeri da 0 a 9), in una forma compatibile con la struttura dei calcolatori, che operano in codice binario.

La rappresentazione di un numero in un calcolatore può assumere varie forme, a seconda che il numero sia intero, reale, ecc., e può occupare una quantità più o meno grande di bit, in dipendenza della precisione con cui si vuole rappresentare il numero stesso.

Le principali modalità di rappresentazione dei numeri utilizzate nei calcolatori sono:

1. numeri interi positivi e relativi
2. numeri razionali

2.1.Numeri Interi

I numeri interi che vengono utilizzati dall'uomo e che devono, dunque, essere rappresentati, possono essere senza segno e con segno.

2.1.1.Numeri Interi Positivi

I numeri interi positivi vengono rappresentati attraverso la loro conversione in binario. Le grandezze rappresentabili vanno da 0 fino a 2^N-1, dove N è il numero di bit utilizzati. Quindi, ad

esempio, con 8 bit è possibile rappresentare tutti i numeri da 0 a 255, con 16 bit tutti i numeri da 0 a 65535.

L'algoritmo di conversione dei numeri interi positivi in binari, è il seguente:

- si divide il numero da convertire per 2
- si riporta la parte intera ottenuta dalla divisione ed il resto della divisione (che può essere 1 o 0 a secondo se il numero diviso per 2 è pari o dispari)
- si ripete il procedimento fino a quando la parte intera ottenuta dalla divisione è zero
- quando il procedimento prima descritto è concluso, il numero binario è costituito da tutti i resti, presi in ordine inverso (ossia da quello ottenuto per ultimo a quello ottenuto per primo)

Esempio: si converta in binario il numero intero positivo 8

8		
4	0	bit meno significativo (LSB)
2	0	
1	0	
0	1	bit più significativo (MSB)

Il numero binario è: 1000

Dato un numero binario, il bit più a destra viene chiamato bit meno significativo (LSB), mentre il bit più a sinistra prende il nome di bit più significativo (MSB).

La conversione opposta, ossia da binario ad intero positivo, si effettua partendo dal bit meno significativo a quello più significativo, moltiplicando ogni bit per la potenza di 2 avente per esponente la posizione del bit. Dato che la posizione del bit meno significativo è 0 mentre la posizione del bit più significativo è N-1 (se N è il numero di bit complessivi), l'esponente delle potenze di 2 è pari a 0 per il bit meno significativo e vale N-1 per il bit più significativo. Tutte le potenze così ottenute, moltiplicate ciascuna per il bit corrispondete, vengono poi sommate. Tutto ciò può essere espresso attraverso la seguente formula:

$$d_{N-1} \cdot 2^{N-1} + d_{N-2} \cdot 2^{N-2} + d_{N-3} \cdot 2^{N-3} + ... + d_1 \cdot 2^1 + d_0 \cdot 2^0$$

dove N è il numero di bit, d_i è il bit di posizione i-esima, assumendo che il bit meno significativo ha posizione 0 e il bit più significativo ha posizione N-1.

Dalla formula precedente si comprende che il più piccolo numero codificabile è 0, mentre il più grande numero è

$$2^{N-1} + 2^{N-2} + 2^{N-3} + ... + 2^1 + 2^0 = 2^N - 1$$

Esempio: si voglia riconvertire il numero binario precedentemente ottenuto dalla divisione, ossia Il numero binario è: 1000101011010101.

Il numero binario è composto da 16 cifre, ossia N=16. In base alla regola di conversione, il numero decimale corrispondente si ottiene applicando la formula:

$$d_{15} \cdot 2^{15} + d_{14} \cdot 2^{14} + d_{13} \cdot 2^{13} + ... + d_1 \cdot 2^1 + d_0 \cdot 2^0 =$$

$1*2^{15}+0*2^{14}+0*2^{13}+0*2^{12}+1*2^{11}+0*2^{10}+1*2^9+0*2^8+1*2^7+1*2^6+0*2^5+1*2^4+0*2^3+1*2^2+0*2^1+1*2^0$

$=32768+2048+512+128+64+16+4+1=35541$

Molto spesso per risparmiare spazio, invece della notazione binaria viene usata quella esadecimale (ossia su base 16). La conversione da binario a esadecimale è immediata: occorre semplicemente dividere il numero binario (formato sempre da multipli interi di 8 bit) in gruppi di 4 bit, partendo dal bit meno significativo. Ad ogni gruppo viene sostituita la corrispondente cifra esadecimale, ricavabile dalla seguente tabella:

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

Esempio: si rappresenti il numero binario 1000101011010101 in esadecimale.

Il precedente numero puo' essere decomposto nei seguenti gruppi da 4 bit, partendo dal bit meno significativo:

0101, che corrisponde nella tabella a 5

1101, che corrisponde nella tabella a D

Rappresentazione dell'informazione

1010, che corrisponde nella tabella a A

1000, che corrisponde nella tabella a 8

Dunque il numero in esadecimale e' 8AD5.

2.1.2.Numeri Interi Relativi

I numeri interi relativi possono essere rappresentati in due modi: ***modulo e segno*** e ***complemento a due.*** Nel seguito entrambe le rappresentazioni verranno illustrate. Verrà dimostrato che la rappresentazione più utilizzata è quella in complemento a due, spiegandone le ragioni.

Rappresentazione in Modulo e Segno.

La rappresentazione per modulo e segno è la più semplice forma di rappresentazione di un numero intero relativo. In accordo a tale rappresentazione, i numeri interi relativi, ossia con segno, vengono rappresentati in modo analogo a quanto fatto per i numeri interi positivi, riservando un bit (tra gli N disponibili) per rappresentare il segno. Generalmente viene scelto il bit più significativo per rappresentare il segno. Se tale bit vale 1 allora il segno rappresentato vale -, se il bit è 0 allora il segno vale +.

E' chiaro che, in base alla precedente scelta, il numero di bit utili per rappresentare il valore assoluto del numero intero relativo è pari a N-1. Dunque se dispongo di N bit, è possibile rappresentre numeri interi relativi il cui valore assoluto (ossia senza segno) è compreso tra 0 e $(2^{N-1}-1)$, ossia l'intervallo dei numeri sarà:

$$-(2^{N-1}-1)\ldots\ldots+(2^{N-1}-1)$$

Dunque se dispongo di 8 bit, è possibile rappresentare numeri interi relativi il cui valore assoluto (ossia senza segno) è compreso tra 0 e 127. Se dispongo di 16 bit, posso rappresentare numeri interi relativi il cui valore assoluto (ossia senza segno) è compreso tra 0 e 32767.

Esempio: si voglia convertire il numero 105 con 8 bit

Il primo bit (quello più significativo) viene posto a 0 perchè il numero è positivo, mentre gli altri 7 bit si calcolano con il metodo delle divisioni per 2, visto prima, applicato al numero 105, ottenendo 1101001. Dunque il numero binario che rappresenta 105 si ottiene aggiungendo ai bit 1101001 il bit di segno 0, messo nella posizione del bit più significativo, ottenendo: 01101001

Esempio: si voglia convertire il numero -105 con 8 bit

Il primo bit (quello più significativo) viene posto a 1 perchè il numero è negativo, mentre gli altri 7 bit si calcolano con il metodo visto prima applicato al numero 105, ottenendo 1101001. Dunque il numero binario che rappresenta 105 è: 11101001

Rappresentazione in Complemento a Due.

Prima di spiegare la rappresentazione di un numero intero relativo in Complemento a 2, è necessario introdurre alcuni concetti sul complemento a 2.

Dato un numero binario di N bit, il complemento a 2 di tale numero si ottiene tramite il seguente algoritmo:

- si procede dal bit meno significativo verso quello più significativo
- se si incontrano tutti bit 0, essi vengono lasciati inalterati
- se si incontra il primo bit 1anche esso viene lasciato inalterato
- tutti i bit successivi al primo bit 1, vengono invertiti (0 diviene 1, e viceversa)

Esempio: si determini il complemento a 2 del numero 10100.

Tutti i bit 0 a partire dal bit meno significativo sono lasciati inalterati e così anche il primo bit 1. Tutti gli altri bit vengono invertiti, ottenendo: 01100.

La rappresentazione in complemento a 2 di un numero intero relativo, si effettua nella seguente maniera:

- i numeri interi positivi sono rappresentati in modulo e segno (con il bit di segno = 0)
- i numeri interi negativi sono rappresentati realizzando il complemento a 2 del numero intero in valore assoluto

Esempio: si voglia convertire il numero 105 con 8 bit

Essendo il numero positivo, la sua rappresentazione in complemento a 2 coincide con quella in modulo e segno, pari a 01101001

Esempio: si voglia convertire il numero -105 con 8 bit

Essendo il numero negativo, la sua rappresentazione in complemento a due si ottiene determinando il complemento a 2 del numero binario corrispondente al valore assoluto (105), ossia di 01101001. Il complemento a 2 è 10010111, che è la rappresentazione di -105.

Si noti che, dato un numero in complemento a 2, la sua conversione in decimale deve avvenire tramite la formula:

$$-d_{N-1}\cdot 2^{N-1}+d_{N-2}\cdot 2^{N-2}+d_{N-3}\cdot 2^{N-3}+....+d_1\cdot 2^1+d_0\cdot 2^0$$

differente da quella utilizzata per la conversione di un numero binario modulo e segno in decimale. Da questa formula si vede che il numero più piccolo che può essere rappresentato con N bit è:

$$-2^{N-1}$$

mentre il numero più grande è:

$$2^{N-2}+2^{N-3}+....+2^1+2^0=2^{N-1}-1$$

Dunque se dispongo di N bit, è possibile rappresentare numeri interi relativi il cui intervallo sarà:

$$-(2^{N-1}).....+(2^{N-1}-1)$$

Da notare che rispetto alla codifica in modulo e segno l'intervallo dei valori è più grande in quanto comprende il valore -2^{N-1}.

Ad esempio se N=16, il numero più piccolo rappresentabile è -2^{15}=-32.768, mentre quello più grande è $2^{15}-1$=+32.767.

Perchè la Rappresentazione in Complemento a 2 è più Conveniente ?

La rappresentazione in complemento a 2 viene utilizzata maggiormente rispetto quella modulo e segno per differenti motivi. Il più rilevante è relativo ai vantaggi ottenibili nell'esecuzione di operazioni elementari come la somma e la sottrazione. Queste due operazioni sono quelle che vengono più frequentemente realizzate in un computer, e, dunque, un risparmio nel tempo necessario alla loro esecuzione comporta un indiscusso aumento delle prestazioni di un computer. Allo scopo di vedere perchè la rappresentazione in complemento a 2 è più vantaggiosa in termini di operazioni di addizione e di sottrazione, nel seguito verrà mostrato come tali operazioni vengono eseguite relativamente alle due rappresentazioni. Prima verranno forniti alcuni concetti sulle operazioni di somma e sottrazione tra numeri binari.

Addizioni e Sottrazioni tra numeri binari.

Le regole per realizzare l'addizione sono le seguenti. Dati due numeri binari, la loro somma è ottenuta applicando le seguenti regole ai bit di medesima posizione:

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

- 1+0=1
- 1+1=0 con riporto di 1
-

Esempio: si vogliano sommare i numeri 0001 (1) e 1010(10).

0001+

1010=

———————

1011 (11)

Le regole per realizzare la sottrazione sono le seguenti. Dati due numeri binari, la loro differenza è ottenuta applicando le seguenti regole ai bit di medesima posizione:

- 0-0=0
- 0-1=1 con prestito di 1 dal bit alla sinistra
- 1-0=1
- 1-1=0

Esempio: si vogliano sottrarre i numeri 1010(10) e 0001 (1).

1010-

0001=

———————

1001 (9)

Addizioni e Sottrazioni tra numeri binari in Rappresentazione Modulo e Segno.

Dati due numeri binari in rappresentazione modulo e segno, l'operazione di addizione può avvenire solo se i due segni sono gli stessi, ossia il bit più significativo è per entrambi i numeri uguale a 0 o a 1. In tal caso la somma tra i due numeri con stesso segno viene realizzata con le regole dell'addizione applicate a tutti i bit tranne il bit di segno. Il numero binario risultante sarà ottenuto aggiungendo il bit di segno ai bit ottenuti dalla somma.

Esempio: Siano dati i numeri a 4 bit 0010 (+2) e 0011 (+3). Si voglia determinare la somma.
Essendo entrambi i numeri positivi (bit più significativo pari a 0), si applichino le regole dell'addizione ai bit:

010+

011=

101 (5)

Il numero binario risultante sarà ottenuto aggiungendo il bit di segno, ossia 0101 (+5)
Dati due numeri binari in rappresentazione modulo e segno, l'operazione di sottrazione può avvenire solo se i due segni sono diversi. In tal caso la sottrazione tra i due numeri viene realizzata con le regole della sottrazione applicate a tutti i bit tranne il bit di segno, sottraendo il numero più piccolo in valore assoluto al numero più grande in valore assoluto. Il numero binario risultante sarà ottenuto aggiungendo ai bit ottenuti dalla sottrazione il bit di segno del numero in valore assoluto più grande.

Addizioni e Sottrazioni tra numeri binari in Rappresentazione in Complemento a Due.

Dati due numeri binari in complemento a due, sia l'operazione di addizione che quella di sottrazione avviene semplicemente applicando le regole dell'addizione a tutti i bit compreso il bit di segno.
Esempio: Siano dati i numeri a 4 bit 0010 (+2) e 1010 (-6). Si voglia determinare la sottrazione.
A tal fine basta applicare semplicemente le regole dell'addizione ai bit:

0010+
1010=

1100 (-4)

Il numero binario risultante è già il risultato son il segno giusto.

Confronto tra Addizioni e Sottrazioni tra numeri binari in Rappresentazione in Modulo e Segno e in Complemento a Due.

Da quanto detto è evidente la semplificazione nel calcolo della somma e della sottrazione dei numeri rappresentati in complemento a due.

2.1.3. Riconoscimento Automatico dell'Overflow del Segno e dello Zero

Una volta compresa la convenienza nell'eseguire tutte le operazioni in complemento a due, è necessario comprendere come il calcolatore possa riconoscere in modo immediato alcuni eventi occorsi a seguito dell'esecuzione dell'operazione di somma in complemento a due.
In molti casi è necessario che il calcolatore capisca subito se una operazione di somma in complemento a due abbia fornito come risultato un numero zero, positivo o negativo. E' chiaro che

il calcolatore potrebbe ottenere queste informazioni effettuando un'operazione di confronto tra il risultato ottenuto e lo zero. Ma è ovvio, altresì, che ciò comporterebbe una grossa perdita di tempo.

Inoltre ci sono casi in cui il calcolatore deve capire se il risultato che è stato ottenuto sia valido o meno. Un risultato non valido si ottiene quando si sommano due numeri tali da produrre un numero più grande del massimo numero codificabile con il numero di bit disponibili. In tal caso il risultato che è stato ottenuto deve essere immediatamente riconosciuto come errato dal calcolatore e dunque scartato.

Al fine di comprendere meglio questo ultimo concetto, si considerino i seguenti esempi.

Si supponga di lavorare a 4 bit (il massimo numero positivo codificabile in complemento a due è 2^3-1, ossia +7, mentre il massimo numero negativo codificabile in complemento a due è -2^3, ossia –8) e si consideri la seguente operazione di somma in complemento a due: si sommino i numeri 1001 (-7) e 1111 (-1). E' chiaro che la somma è -8, ossia il limite inferiore codificabile con 4 bit, tramite la codifica in complemento a due. Eseguendo la somma, il calcolatore ottiene:

1001+

1111=

_____

1 1000

dove l'1 a sinistra, ottenuto come resto, viene perso per superamento della capacità dei registri. Il numero 1000 in complemento a 2 significa –8, che rappresenta il risultato corretto.

Si supponga ancora di lavorare a 4 bit e si consideri la seguente operazione di somma in complemento a due: si sommino i numeri 1001 (-7) e 1110 (-2). E' chiaro che la somma è -9, non codificabile con 4 bit. Eseguendo la somma, il calcolatore ottiene:

1001+

1110=

_____

1 0111

dove l'1 a sinistra, ottenuto come resto, viene perso per superamento della capacità dei registri.

Il numero 0111 in complemento a 2 significa +7, che rappresenta un risultato errato.

Si supponga adesso di lavorare ad 8 bit (il massimo numero positivo codificabile in complemento a due è 2^7-1, ossia +127, mentre il massimo numero negativo codificabile in complemento a due è -2^7, ossia –128) e si consideri la seguente operazione di somma in complemento a due: si sommino i numeri 01111110 (126) e 00000011 (3). E' chiaro che la somma è 129 superiore al numero massimo positivo codificabile con 8 bit, tramite la codifica in complemento a due. Eseguendo la somma, il calcolatore ottiene:

01111110+
00000011=

———————
10000001

Il numero 10000001 in complemento a 2 significa –128+1=-127, che rappresenta palesemente un risultato errato.

In base agli esempi appena descritti, nasce il seguente problema: come fa il calcolatore a capire che il risultato di una somma in complemento a 2 è un risultato errato o meno ?

Al fine di poter comprendere immediatamente se il risultato di un'operazione di somma in complemento a due è errata, e, nel caso in cui il risultato sia corretto, se il risultato sia positivo, negativo o uguale a zero, il calcolatore utilizza tre particolari flag: Overflow Flag (OF), Sign Flag (SF) e Zero Flag (ZF). Il valore assunto da tali flag alla fine dell'operazione di somma viene stabilito con le seguenti regole:

Siano X e Y i due numeri in complemento a due da sommare. Sia S il risultato ottenuto, alla fine dell'operazione di somma i valori dei flag sono:

- OF=1, ossia il risultato S non è valido, se i bit più significativi di X e Y sono uguali e il bit più significativo di S è diverso da essi.
- SF=1, ossia il risultato che deve essere valido è negativo, se il bit più significativo di S è uguale a 1
- ZF=1, ossia il risultato che deve essere valido è zero, se S = 0.

Esempio. Si sommino i numeri 01111110 (126) e 00000011 (3). Eseguendo la somma, il calcolatore ottiene:

01111110+
00000011=

———————
10000001

- OF=1, ossia il risultato S non è valido, perché i bit più significativi di X e Y sono uguali e il bit più significativo di S è diverso da essi.
- SF non viene considerato perché il risultato non è valido.
- ZF non viene considerato perché il risultato non è valido.

2.2.Numeri Reali

I numeri reali possono essere rappresentati in due modalità: virgola fissa e virgola mobile.

2.2.1.Rappresentazione in Virgola Fissa.

La rappresentazione in virgola fissa consiste nel rappresentare un numero reale con segno tramite N bit, supponendo fissa la posizione della virgola.

In un numero rappresentato in virgola fissa a N bits, viene utilizzato un bit per il segno, I bits per rappresentare la parte intera e D bits per rappresentare la parte decimale (ovviamente sarà N = I + D +1).

Si consideri la seguente figura. Come visibile il bit più significativo è stato utilizzato per il segno, i primi D bits (quelli meno significativi) per la parte frazionaria e i rimanenti bits (in numero di I) per rappresentare la parte intera.

+/-	I-1	I-2	I-3	....	0	-1	-2	...	-D
Segno	Parte Intera					Parte Frazionaria			

Per convertire un numero rappresentato in Virgola Fissa con 1 bit di segno (s), I bits di parte intera e D bits di parte decimale:

$$N_2 = s\ a_{I-1}a_{I-2}a_{I-3}.....a_1a_0b_{-1}b_{-2}.....b_{-D}$$

basta applicare la seguente formula:

$$N_{10} = (-1)^s \cdot \left(\sum_{i=0}^{I-1} a_i \cdot 2^i + \sum_{d=-1}^{-D} b_d \cdot 2^d\right)$$

Esempio. Convertire in decimale il numero binario in virgola fissa 1010,101.

- Segno: 1
- Parte Intera: $2^1 = 2$
- Parte Frazionaria: $2^{-1} + 2^{-3} = 0.5 + 0.125 = 0.625$

Quindi il numero decimale corrispondente a 1010,101 è -2,625.

Per convertire un numero decimale in rappresentazione binaria in virgola fissa, si opera come segue. La parte intera si codifica in binario, con il segno. La parte frazionaria si converte così:

- si moltiplica per 2 e nel risultato si separa la parte intera (che è sempre 0 o 1);
- si ripete il procedimento fino a quando il risultato della moltiplicazione è 1.000... oppure si raggiunge il numero di cifre binarie dedicate alla codifica della parte frazionaria.

Il limite della rappresentazione in virgola fissa risiede proprio nella rigidità della posizione assegnata alla virgola. Infatti è stato visto che nella rappresentazione in Virgola Fissa, il numero di bits assegnati per codificare la parte intera e quelli assegnati per codificare la parte frazionaria sono fissi. Quindi, se per esempio si fissa la virgola in modo tale che la gran parte dei bits è dedicata per codificare la parte intera, allora la precisione nel codificare numeri piccoli sarà molto bassa.

2.2.2.Rappresentazione in Virgola Mobile.

In un numero rappresentato in virgola mobile vengono stabiliti un certo numero di bit assegnati per codificare il segno (s), la mantissa (m) ed un certo numero di bit per codificare l'esponente (e).
A titolo di esempio si consideri la seguente rappresentazione. Il bit più significativo è il bit del segno (s), mentre l'esponente è stato rappresentato con 7 bits (e) e la mantissa con i restanti 24 bits (m).

s (31 bit)	e(30...24 bit)	m(23...0 bit)

- Il numero occupa in tutto 32 bit.
- La mantissa è rappresentata secondo la codifica binaria su 24 bits.
- L'esponente è rappresentato in codifica binaria su 7 bits.
- Il bit s, determina il segno del numero reale in base al suo valore (1-numero negativo, 0-numero positivo)

In teoria esistono infinite possibilità di codificare un numero reale in virgola mobile, basti pensare alle innumerevoli scelte del numero di bit per codificare l'esponente e di quelle per codificare la mantissa.
In realtà dal 1985 esiste una regola di codifica dei numeri reali in virgola mobile, nota come standard IEEE 754. Essa è adottata da tutti i programmi di compilazione.

Lo standard 754 dell'IEEE

Nell'anno 1985 l'IEEE (The Institute of Electrical and Electronics Engineering) ha definito uno standard per la codifica dei numeri reali in virgola mobile, che verrà illustrato nel seguito. Lo standard prevedeva inizialmente due codifiche a 32 (float) e a 64 bit (double). Esistono poi delle estensioni, ad esempio quella a 80 bits.
Nella codifica float a 32 bit, vengono assegnati:

- 1 bit per il segno (il bit più significativo), s

Rappresentazione dell'informazione

- 8 bit per l'esponente, e
- 23 bit per la mantissa, m

Nella codifica double a 64 bit, vengono assegnati:

- 1 bit per il segno (il bit più significativo), s
- 11 bit per l'esponente, e
- 52 bit per la mantissa, m

La rappresentazione IEEE 754 prevede due diverse forme: quella normalizzata e quella denormalizzata. Si usa generalmente quella normalizzata.

Consideriamo dapprima la forma normalizzata.

Nella forma normalizzata si ha che il numero decimale corrispondente è dato dalla formula:

dove: $$N_{10} = (-1)^s \cdot f \cdot 2^{esp-p}$$

- e≠0000…..0 (che è la codifica riservata per la forma denormalizzata), e≠1111….1 (che è la codifica riservata per il simbolo di infinito)
- esp è la conversione binaria senza segno della porzione di bit relativi all'esponente e. Si ha che 1≤esp≤max, dove max=254 (float), max=2046 (double)
- p=polarizzazione (bias), p=127 (float), p=1023 (double)
- f è la codifica in virgola fissa, considerando m come parte frazionaria e assumendo sempre che la parte intera sia 1.

Si considerino i seguenti esempi:

- Si converta il numero binario 1 10000001 01000000000000000000000
  - Segno negativo –
  - esp-p=2^7+2^0-127=129-127=2
  - f=$1+2^{-2}$=1.25
  - Numero=$-1.25*2^2$=-5
- Si converta il numero binario 0 10000011 10011000000000000000000
  - Segno positivo +
  - esp-p=$2^7+2^1+2^0$-127=131-127=4
  - f=$1+2^{-1}+2^{-4}+2^{-5}$=1.59375
  - Numero=$1.59375*2^4$=25.5

Il numero reale più piccolo in valore assoluto che si può ottenere dalla codifica in virgola mobile normalizzata è quando la codifica binaria della mantissa è composta da tutti 0 e l'esponente "e" assume il valore minimo, ossia 00000.1, ovvero esp=1; nel caso di float si ha:

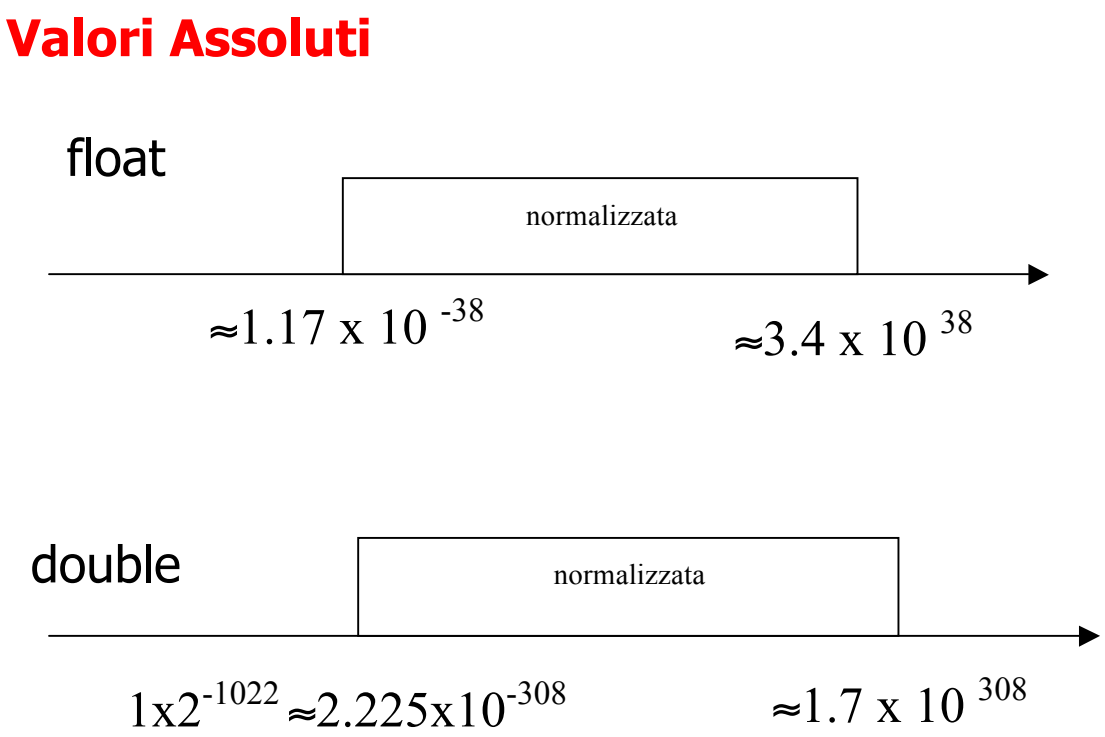
Rappresentazione dell'informazione

$$N_{\min} = 1.(000000000....0)_2 \cdot 2^{1-127} = 1.0 \cdot 2^{-126}$$

mentre il numero più grande si ottiene quando la codifica binaria della mantissa prevede una sequenza di 1 e l'esponente assume il valore massimo, ossia 254; nel caso di float si ha:

$$N_{\max} = 1.(11111111...1)_2 \cdot 2^{254-127} = 1.(11111111...1)_2 \cdot 2^{127} \approx 2 \cdot 2^{127} \approx 2^{128} \approx 3{,}4 \cdot 10^{38}$$

La seguente figura riporta gli estremi dei valori assoluti nel caso di float e di double:



Nella codifica IEEE 754 esistono tre valori particolari rappresentabili, ossia:

- Infinito, ∞, quando e=11111.1, esp=255 e m=000000.0
- NaN (Not a Number, ossia valore indefinito) quando e=1111.1, esp=255 e m≠0. Quest'ultimo valore è utilizzato nel caso di operazioni che forniscono un risultato indefinito (ad esempio le divisioni 0/0 e ∞/∞).
- Zero, quando e=00000.0, e m=00000.0.

La forma denormalizzata, si ha quando l'esponente è uguale a 0. In tal caso la mantissa può assumere una qualunque sequenza di bit tranne la sequenza di tutti 0 (ossia da 000000..1 fino a 1111111...1). La codifica dei numeri reali nella forma denormalizzata è:

dove:

$$N_{10} = (-1)^s \cdot f \cdot 2^{-p}$$

- p=polarizzazione (bias), p=126 (float), p=1022 (double)

Rappresentazione dell'informazione

- f = si utilizza la codifica in virgola fissa, considerando m come parte frazionaria e assumendo sempre che la parte intera sia 0

Il numero reale più piccolo in valore assoluto che si può ottenere con la forma denormalizzata a 32 bits è:

X 00000000 00000000000000000000001 (X può essere 0 o 1)

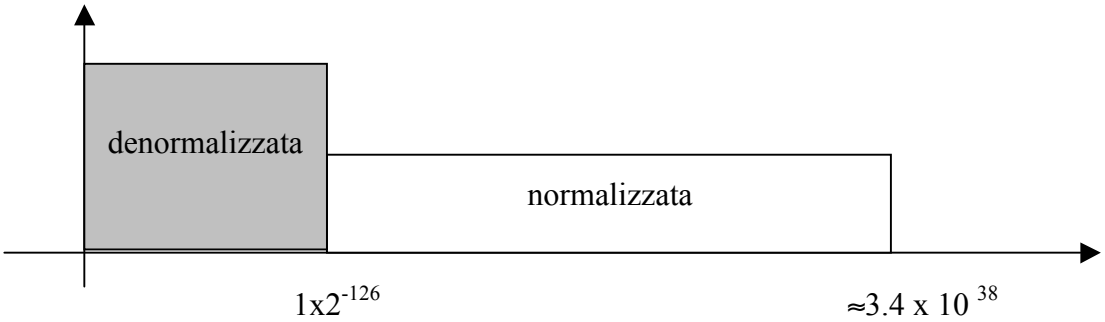
- $f=0+2^{-23}$
- $Numero=\pm 2^{-23}*2^{-126}= \pm 2^{-149} \sim \pm 1.4*10^{-45}$

mentre il numero più grande è:

X 00000000 11111111111111111111111

- $f=0+2^{-1}+2^{-2}+2^{-3}+\ldots+2^{-23}\sim 1$
- $Numero\sim\pm 1*2^{-126} \sim\pm 1.17*10^{-38}$

La seguente figura mostra gli intervalli di codifica del valore assoluto di un numero reale nelle due forme normalizzata e denormalizzata, considerando il float:



La seguente figura mostra gli intervalli di codifica del valore assoluto di un numero reale nella forma normalizzata, considerando la rappresentazione double:

La forma denormalizzata ha il vantaggio di rappresentare con estrema precisione numeri piccoli attorno allo zero. Il principale svantaggio è che richiede algoritmi più complicati per l'esecuzione di calcoli aritmetici.

Per quanto riguarda le regole di conversione di un numero decimale in un numero binario in virgola mobile IEEE 754, nel seguito verrà descritta la procedura considerando la forma normalizzata.

Rappresentazione dell'informazione

Dato un numero reale per ottenere la sua rappresentazione binaria in virgola mobile in **forma normalizzata** secondo lo standard IEEE 754, si procede nel seguente modo:

1. si considera il numero senza segno. Il bit di segno s viene posto a 0 se il numero è positivo, e ad 1 se è negativo;
2. si considera la parte intera del numero e si converte in binario;
3. si considera la parte frazionaria ottenuta sottraendo al numero reale, la parte intera. La parte frazionaria si converte in maniera simile a quanto descritto per i numeri in virgola fissa, ossia:
    - si moltiplica per 2 e nel risultato si sottrae la parte intera (che è sempre 0 o 1);
    - si ripete il procedimento fino a quando il risultato della moltiplicazione è 1.000, sottraendo l'1.
4. si considera il numero binario ottenuto convertendo la parte intera e la parte frazionaria.
5. Si normalizza il numero binario ottenuto al passo precedente. La normalizzazione si ottiene spostando la virgola verso sinistra di p posizioni fino ad ottenere il numero binario espresso nella forma normalizzata (ossia 1,xxxxxx). Il numero binario ottenuto viene moltiplicato per la potenza di 2 elevato a p. La mantissa m è la sequenza di bit dopo la virgola, codificata in 23 o 52 bits a seconda della rappresentazione float o double. Ad esempio:
    - Si supponga che 101,1 sia il numero binario ottenuto al passo 4
    - il numero 101,1 è equivalente alla forma normalizzata $1{,}011 \times 2^{2}$, perché $101{,}1 = 1x2^{2}+0x2^{1}+1x2^{0}+1x2^{-1}$ (si veda la codifica in virgola fissa). Ma $1{,}011x2^{2} = (1x2^{0}+0x2^{-1}+1x2^{-2}+1x2^{-3}) \times 2^{2} = 1x2^{2}+0x2^{1}+1x2^{0}+1x2^{-1}$, e dunque le due espressioni sono identiche. La mantissa m è dunque pari a 011. La mantissa in rappresentazione float su 23 bits è 01100000000000000000000.
6. Si calcola l'esponente "e" sommando a p il numero 127 o 1023 a seconda del tipo di rappresentazione (float o double). Si codifica in binario su 8 o 11 bits a seconda della codifica float o double, l'esponente trovato. Ad esempio in float:
    - se p=2, allora l'esponente e=2+127=129, la cui codifica su 8 è 10000001
7. Si mettono assieme segno (s), esponente (e) e mantissa (m) trovati ai passi precedenti, secondo lo schema di rappresentazione della virgola mobile.

Esempio. Si converta il numero N=7,5 nella rappresentazione binaria IEEE 754 in virgola mobile float. Si procede nel seguente modo:

1. si converte la parte intera 7 = 111

2. si considera la parte frazionaria 0,5:
    - 0,5 x 2 = 1,0
3. si considera il numero binario ottenuto convertendo la parte intera e la parte frazionaria: 111,1
4. Si normalizza il numero binario ottenuto al passo precedente: $1{,}111 \times 2^2$.
5. e=2+127=129, la cui codifica su 8 bits è 10000001
6. s=0, e=10000001, m=11100000000000000000000

3.Codici Alfanumerici

L'evoluzione dei calcolatori li ha portati a diventare, oltre che elaboratori di numeri, anche elaboratori di altri tipi di informazione, prima fra tutti quella testuale. I simboli che vengono usati per rappresentare testi sono, come noto, i caratteri alfanumerici, cioè l'insieme costituito dalle lettere dell'alfabeto e dalle dieci cifre decimali. A questi vanno aggiunti diversi altri simboli come lo spazio, i segni di interpunzione, i simboli per indicare il passaggio alla riga o alla pagina successiva, ecc.

Questo insieme di caratteri alfanumerici può essere facilmente rappresentato attribuendo in maniera univoca a ciascuno dei suoi elementi un numero intero (codice). Osserviamo che il numero delle lettere dell'alfabeto inglese sono 26, per un totale di 52, considerando anche quelle maiuscole. Se ad esse aggiungiamo le dieci cifre numeriche, una quarantina di simboli extra, arriviamo ad un totale di un centinaio di simboli da rappresentare. Tale numero suggerisce che soli 7 bit sono sufficienti per rappresentare l'insieme dei caratteri alfanumerici (7 bit permettono di rappresentare 128 simboli diversi).

E' chiaro che è necessario che la rappresentazione dei simboli sia la stessa in tutto il mondo, pena la totale incomunicabilità. E', dunque, necessario l'adozione di una comune rappresentazione. Attualmente esistono due diverse rappresentazioni: codifica ASCII e relative estensioni e codifica Unicode.

3.1.Codifica ASCII

La codifica ASCII (che si pronuncia ASKI), prende il nome da **American Standard Code for Information Interchange**. Tale codifica si basa sull'utilizzo di 7 bit per un totale di 128 simboli rappresentabili. Da notare che i caratteri dell'alfabeto e le cifre numeriche successive hanno codice

anch'esso successivo (ad esempio a ha codice 97, b codice 98, c codice 99, il numero 0 ha codice 48, il numero 1 codice 49, etc.)

Tra le più utilizzate codifiche ASCII (entro i primi 128 simboli) vi sono:

1. ~ (tilde) codice 126
2. { codice 123
3. } codice 125
4. | codice 124

La seguente tabella riporta i caratteri la cui codifica è compresa tra 0 e 127.

	0	1	2	3	4	5	6	7
0	NUL	DLE	Spazio	0	@	P	‘	p
1	SOH	F1	!	1	A	Q	a	q
2	STX	F2	“	2	B	R	b	r
3	ETX	F3	#	3	C	S	c	s
4	EOT	F4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	‘	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	↑	n	~
F	SI	US	/	?	O	←	o	DEL

Come si vede:

- I caratteri di controllo (non riproducibili) hanno i codici più bassi.
- Il blank (Spazio) è il primo dei caratteri riproducibili.
- Le maiuscole/minuscole sono ordinate (codice Progressivo).

3.2.Codifiche derivate dalla codifica ASCII

Esistono numerose estensioni della codifica ASCII. Tali estensioni derivano dalla necessità di codificare simboli legati a particolari lingue, e dal fatto che operando su 8 bit, la codifica ASCII consente l'utilizzo dell'ottavo bit, lasciando gli altri 7 inalterati.

Tutte le estensioni della codifica ASCII non modificano tale codifica ma aggiungono semplicemente altri 128 simboli.

Tra le estensioni più diffuse vi è la ISO Latin 1.