Rappresentazione e operazioni binarie

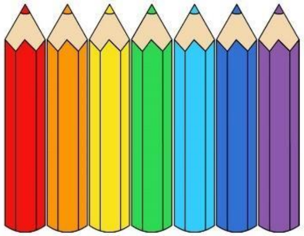
Corso di Architettura degli Elaboratori 1

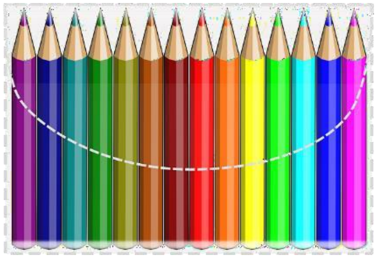
Fulvio Valenza

A.A. 2018-2019

Polo di Vercelli

SISTEMI DI NUMERAZIONE POSIZIONALI





7 ***VII*** ***14*** ***XIV***

- I numeri permettono di rappresentare una **QUANTITÀ** con un **SIMBOLO**
- Alcuni sistemi di numerazione sono POSIZIONALI (per esempio il **sistema decimale** che usiamo normalmente) mentre altri non lo sono (per esempio i numeri romani)

SISTEMI DI NUMERAZIONE POSIZIONALI

- Nei sistemi di numerazione posizionali viene associato un peso a ciascuna posizione (cifra) all'interno della rappresentazione del numero.
- Il valore del numero è ottenuto facendo la somma dei prodotti di ciascuna cifra per il relativo peso

Esempio: numerazionedecimale

$$230 = 2 \cdot 10^2 + 3 \cdot 10^1 + 0 \cdot 10^0$$

Centinaia, Decine, Unità

Rappresentazione posizionale di un numero ***in base 10***

SISTEMI DI NUMERAZIONE POSIZIONALI

- ***Regola generale*** (valida per qualsiasi **base r**)
  Rappresentazione del numero (le cifre d_i sono tra 0 e r-1)

$$N_{(r)} = d_{n-1}d_{n-2} \ldots d_0$$

cifra meno significativa

cifra più significativa

Valore del numero in base r

$$V(N_{(r)}) = r^{n-1} \cdot d_{n-1} + r^{n-2} \cdot d_{n-2} + \ldots + r^0 \cdot d_0$$

Vale anche per i numeri frazionari (con il punto decimale)

$$N_{(r)} = \underbrace{d_{n-1}d_{n-2} \ldots d_0}_{\textit{parte intera}}.\underbrace{d_{-1}d_{-2} \ldots d_{-m}}_{\textit{parte frazionaria}}$$

SISTEMI DI NUMERAZIONE POSIZIONALI

- Formula Generale:

$$V(N_{(r)}) = \sum_{i=-m}^{n-1} \mathrm{r}^{\mathrm{i}}\, d_{\mathrm{I}}$$

ESEMPIO:

$$230.15_{(10)} = 2 \cdot 10^{2} + 3 \cdot 10^{1} + 0 \cdot 10^{0} + 1 \cdot 10^{-1} + 5 \cdot 10^{-2}$$

Centinaia Decine Unità Decimi Centesimi

Rappresentazione in base 2

- **Numeri binari naturali**: si utilizza un sistema di numerazione posizionale in base 2
- La sequenza di bit:

$$b_n \; b_{n-1} \; \dots \; b_0$$

- dove si ha b_i vale **0** oppure **1**, in base 2 rappresenta:

$$b_n * 2^{n-1} + b_{n-1} * 2^{n-2} + b_{n-2} * 2^{n-3} + \dots + b_n * 2^0$$

Esempio:

$$11001_{(2)} = 2^4 \cdot 1 + 2^3 \cdot 1 + 2^2 \cdot 0 + 2^1 \cdot 0 + 2^0 \cdot 1 = 25_{(10)}$$

Potenza del 2

n	2^n	n	2^n	n	2^n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096	20	1,048,576
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

https://play2048.co/

Rappresentazione in base 8

- Le cifre sono 8: {0,1,2,3,4,5,6,7}

- In base 8 la sequenza di pesi associati alle varie cifre è:

$$b_n * 8^{n-1} + b_{n-1} * 8^{n-2} + b_{n-2} * 8^{n-3} + \ldots + b_n * 8^0$$

Esempio:

$$51_{(8)} = 8^1 \cdot 5 + 8^0 \cdot 1 = 41_{(10)}$$

Rappresentazione in base 16

- Le cifre sono 16: {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

- In base 16 la sequenza di pesi associati alle varie cifre è:

$$b_n * 16^{n-1} + b_{n-1} * 16^{n-2} + b_{n-2} * 16^{n-3} + \ldots + b_n * 16^{0}$$

Esempio:

$$5A_{(16)} = 16^1 \cdot 5 + 16^0 \cdot 10 = 90_{(10)}$$

Conversione tra basi

- Da base 10 a base 2 e viceversa
- Da base 10 a base 8 o 16 e viceversa
- Da base 2 a base 8 o 16 e viceversa
- Conversioni Frazioni

Conversione da base 10 a base 2

Metodo 1: Sottrazioni successive

1) Trovare la più grande potenze del 2 non superiore a N

2) Sottrarre tale Potenza al numero N in base 10

3) Ripetere l'operazione con il risultato della sottrazione finché si ottiene 0.

Al termine comporre le cifre binarie inserendo un 1 in corrispondenza delle potenze di 2 sottratte, 0 nelle altre posizioni.

ESEMPIO: $49_{(10)}$

49 -**32** = 17; 17 -**16** = 1; 1-**1** = 0; $49=2^5 + 2^4 + 2^0$

$$\mathbf{49_{(10)} = 110001_{(2)}}$$

Conversione da base 10 a base 2

Metodo 2: Divisioni Successive

1. Dividere il numero N in base 10 per 2: sia Q il quoziente ed R il resto
2. Ripetere l'operazione con il quoziente finché si ottiene 0.
3. Al termine allineare i resti a partire dall'ultimo fino al primo

ESEMPIO: $49_{(10)}$

$49_{(10)} = 110001_{(2)}$

49:2	Q=24	R=**1**
24:2	Q=12	R=**0**
12:2	Q=6	R=**0**
6:2	Q=3	R=**0**
3:2	Q=1	R=**1**
1:2	Q=0	R=**1**

Numero

Conversione da base 10 a base 2

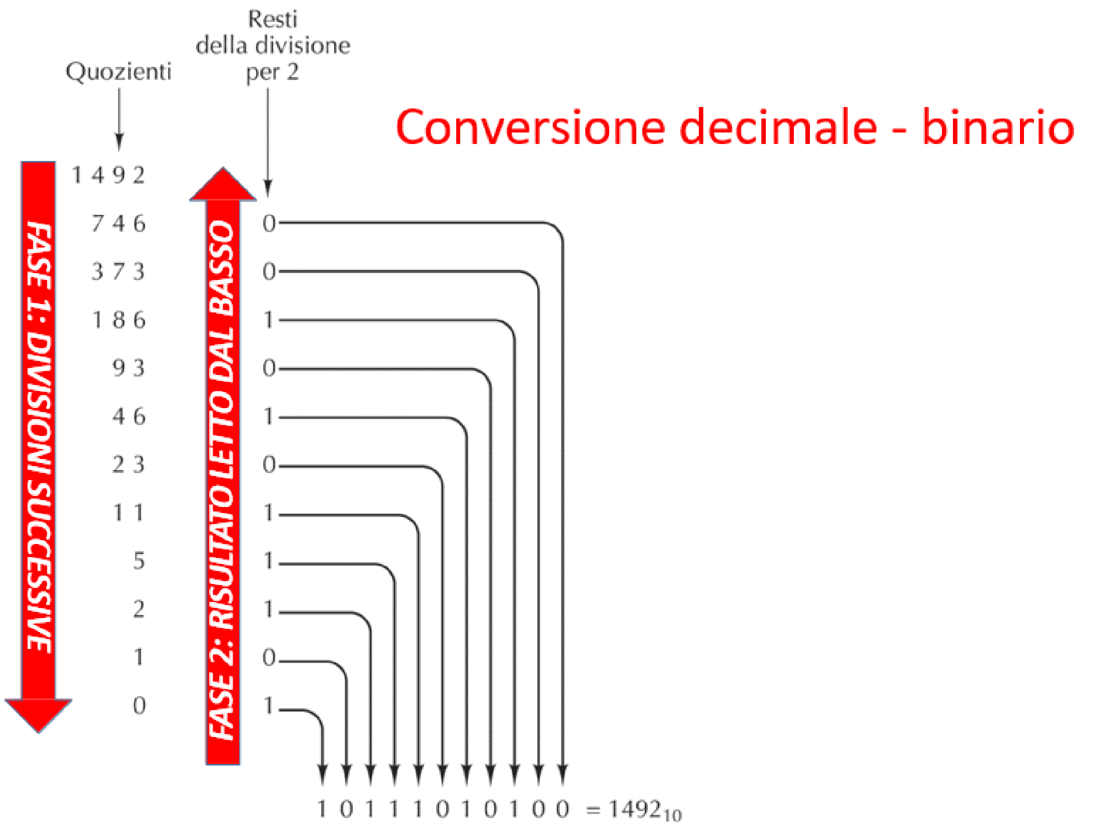


Figura A.5 La conversione del numero decimale 1492 in binario mediante dimezzamenti successivi, partendo dall'alto e procedendo verso il basso. Per esempio, 93 diviso 2 fa 46 con resto 1, riportati nella riga successiva.

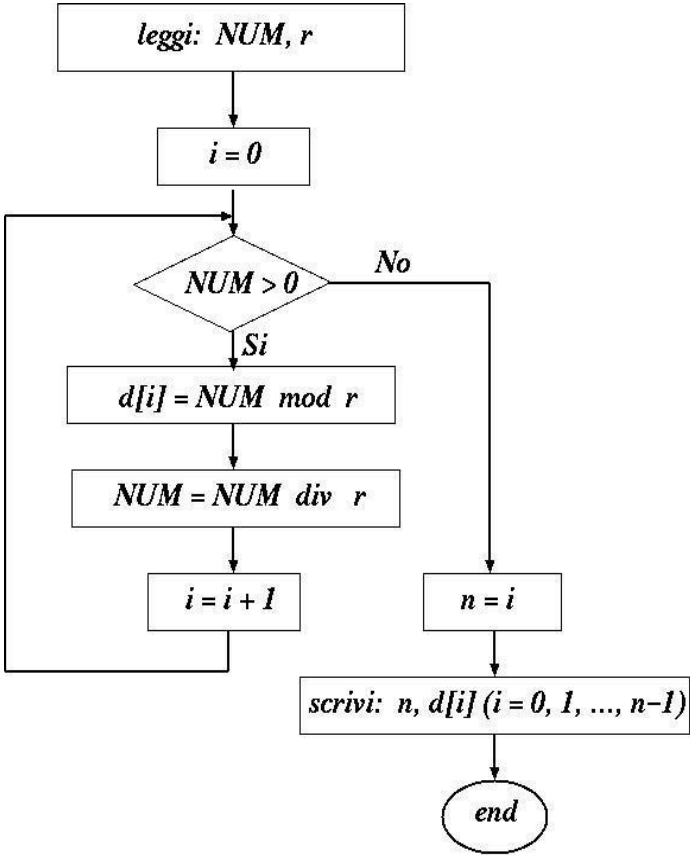
Conversione da base 10 a base r

Algoritmi di trasformazione da base 10 a base r

- **Input**: NUM il numero in base 10 e r la nuova base

- **Output**: il numero n di cifre che servono per rappresentare NUM in base r, e le n cifre in base r

- La cifra d[i] è quella di peso r^i



Esempi Conversione da base 10 a base r

Convertire 25_{10} in binario.

$$
\begin{array}{llllll}
25 & :2 & & & & \\
 & 12 & :2 & & & \text{resto} \Rightarrow 1 \\
 & & 6 & :2 & & \text{resto} \Rightarrow 0 \\
 & & & 3 & :2 & \text{resto} \Rightarrow 0 \\
 & & & & 1 \quad :2 & \text{resto} \Rightarrow 1 \\
 & & & & \quad 0 & \text{resto} \Rightarrow 1
\end{array}
$$

$25_{10} \Rightarrow 11001_2$

Convertire 9852_{10} in ottale.

$$
\begin{array}{llllll}
9852 & :8 & & & & \\
\hline
 & 1231 & :8 & & & \\
\cline{2-3}
 & & 153 & :8 & & \\
\cline{3-4}
 & & & 19 & :8 & \\
\cline{4-5}
 & & & & 2 & :8 \\
\cline{5-6}
 & & & & & 0
\end{array}
$$

resto $\Rightarrow$ 4
resto $\Rightarrow$ 7
resto $\Rightarrow$ 1
resto $\Rightarrow$ 3
resto $\Rightarrow$ 2

$9852_{10} \Rightarrow 23174_8$

Trasformazione da base 2 a base 10

- **Metodo 1**:

Si applica la definizione di numero posizionale: si sommano le potenze di 2 corrispondenti alle posizioni dove nel numero compaiono cifre = 1

ESEMPIO: $11001001_{(2)}$

1	1	0	0	1	0	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$128+64+8+1 = 201_{(10)}$$

Trasformazione da base 2 a base 10

- **Metodo 2:**

Si applicano ripetute moltiplicazioni per 2 e somma delle cifre binarie a partire dalla più significativa

ESEMPIO: $11001001_{(2)}$

Cifra binaria		Somma parziale
1	0*2+1	1
1	1*2+1	3
0	3*2+0	6
0	6*2+0	12
1	12*2+1	25
0	25*2+0	50
0	50*2+0	100
1	100*2+1	$201_{(10)}$

Trasformazione da base 2 a base 10

Conversione binario - decimale

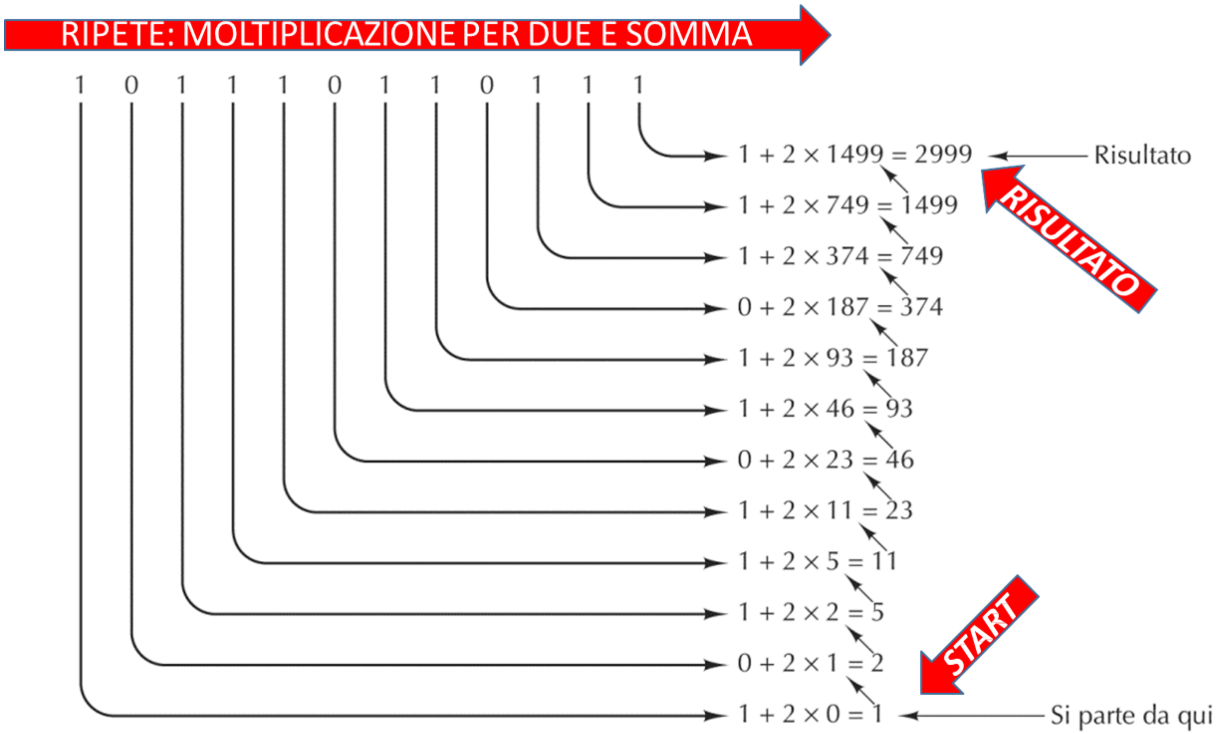
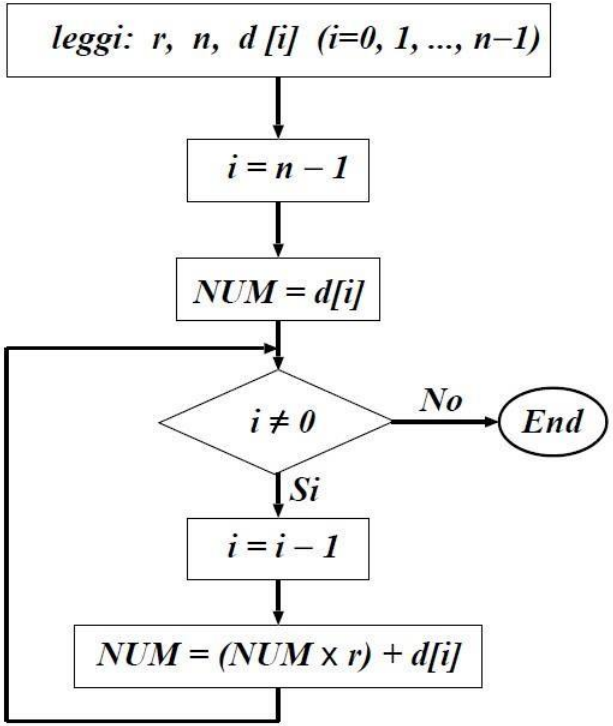


Figura A.6 La conversione del numero binario 101110110111 in decimale mediante raddoppiamenti successivi, a partire dal basso. Ogni riga si ottiene raddoppiando l'elemento della riga precedente e sommandogli il bit corrispondente. Per esempio, 749 è due volte 374 più il bit 1 che si trova in corrispondenza della riga di 749.

Trasformazione da base 2 a base r

Algoritmi di trasformazione da base r a base 10

- **Input**: il numero n di cifre che servono per rappresentare NUM in base r, e le n cifre in base r
- **Output**: NUM il numero in base 10
- La cifra d[i] è quella di peso r^i



Trasformazione da base 10 a base 8 o 16

- Per trasformare un numero da base 10 a base 8 o 16 si possono utilizzare gli stessi metodi visti prima

Esempio: da base 10 a base 8

49:8	Q=6	R=1
6:8	Q=0	R=6

$49_{(10)} = 61_{(8)}$

Esempio: da base 10 a base 16

49:16	Q=3	R=1
3:16	Q=0	R=3

$49_{(10)} = 31_{(16)}$

Trasformazione da 2 a 8/16 e viceversa

- La trasformazione da base 2 a base 8 o 16 si può realizzare facilmente **raggruppando le cifre binarie a gruppi di 3 oppure 4** a partire dalla cifra meno significativa; ogni gruppo viene poi sostituito dalla corrispondente cifra ottale o esadecimale

- Per la trasformazione inversa basta sostituire **ogni cifra ottale o esadecimale con il corrispondente gruppo di 3 o 4 bit**

Trasformazione da 2 a 8/16 e viceversa

ESEMPIO: $11001001_{(2)}$

- Da base 2 a base 8: $\boxed{011}\boxed{001}\boxed{001}_{(2)} = 311_{(8)}$

  cifra aggiunta per formare gruppo da 3 bit

- Da base 2 a base 16: $11001001_{(2)} = C9_{(16)}$

ESEMPIO:

- Da base 8 a base 2:

3	1	1	(8)
011	001	001	(2)

- Da base 2 a base 16:

C	9	(16)
1100	1001	(2)

Codifica binaria di cifre ottali ed esadecimali

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

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

Conversione di numeri con parte frazionaria

- La conversione dei numeri frazionari dev'essere fatta separatamente per la parte intera e per la parte frazionaria.

- Ricordiamo che le cifre della parte frazionaria hanno pesi corrispondenti alle potenze negative della base:

$$N_{(r)} = \underbrace{d_{n-1}d_{n-2}\ \dots\ d_0}_{\textit{parte intera}}.\underbrace{d_{-1}d_{-2}\ \dots d_{-m}}_{\textit{parte frazionaria}}$$

Conversione frazionaria da 10 a 2

- Il procedimento potrebbe NON TERMINARE, infatti non è detto che un numero in base 10 con un numero finito di cifre dopo la virgola sia esattamente rappresentabile con un numero finito di cifre in base 2. In questi casi si stabilisce la precisione che si vuole mantenere e si interrompe il procedimento subito dopo la generazione della cifra corrispondente alla precisione desiderata.

Conversione frazionaria da 10 a 2

- Descrizione dell'algoritmo tramite esempio:

$$
\begin{array}{lcll}
0.3125 \cdot 2 & = & 0.625 & \Rightarrow \text{bit } 0 \\
0.625 \cdot 2 & = & 1.25 & \Rightarrow \text{bit } 1 \\
0.25 \cdot 2 & = & 0.5 & \Rightarrow \text{bit } 0 \\
0.5 \cdot 2 & = & 1.0 & \Rightarrow \text{bit } 1 \\
0 \cdot 2 & = & 0 &
\end{array}
$$

Numero

$0.3125_{(10)} = 0.0101_{(2)}$

$$
\begin{array}{lcll}
0.2 \cdot 2 & = & 0.4 & \Rightarrow \text{bit } 0 \\
0.4 \cdot 2 & = & 0.8 & \Rightarrow \text{bit } 0 \\
0.8 \cdot 2 & = & 1.6 & \Rightarrow \text{bit } 1 \\
0.6 \cdot 2 & = & 1.2 & \Rightarrow \text{bit } 1 \\
0.2 \cdot 2 & = & \ldots &
\end{array}
$$

Numero

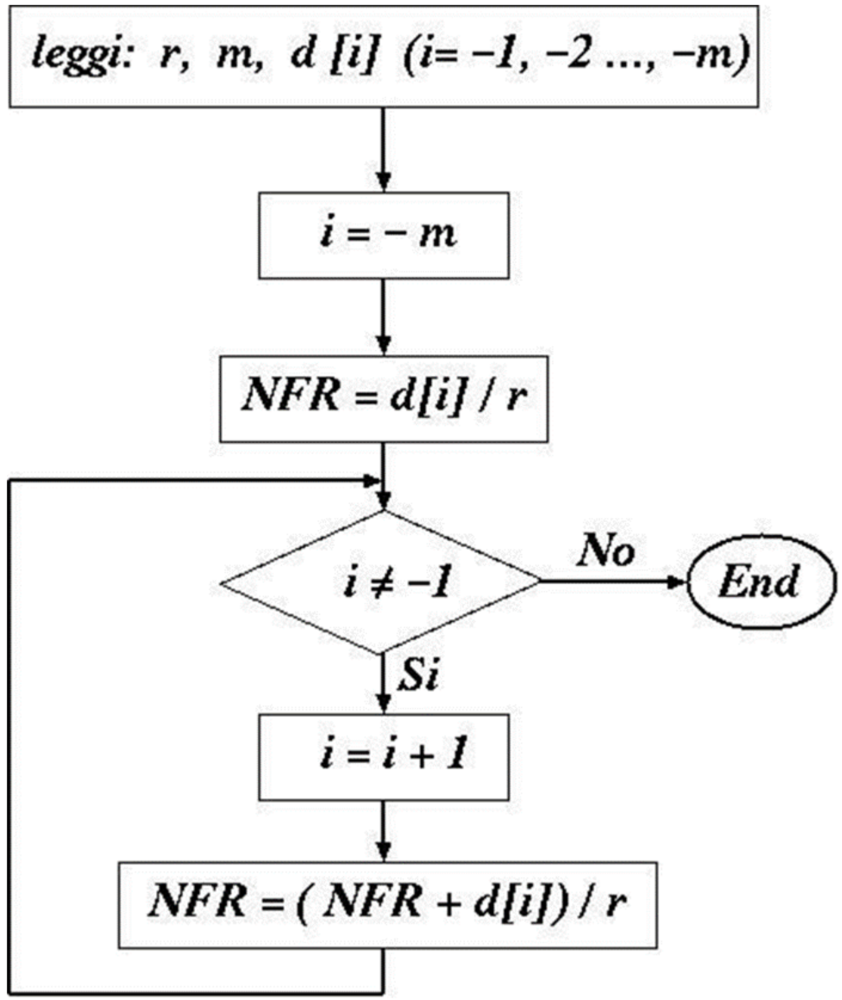
$0.2_{(10)} = 0.0011_{(2)}$

Conversione frazionaria da 10 a 2

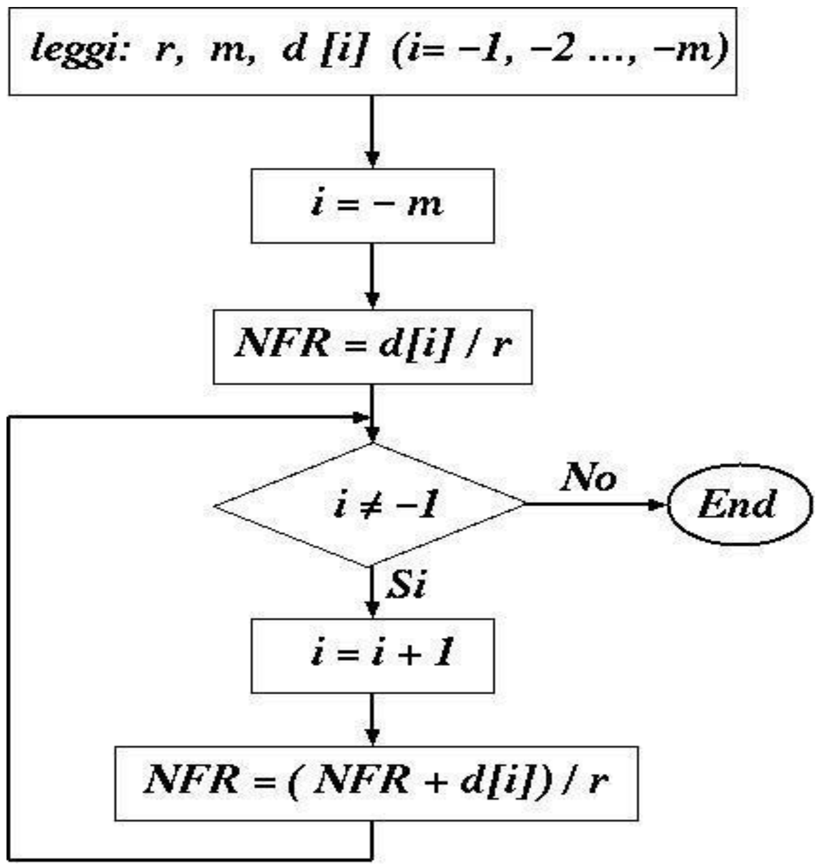
- *Pesi:*

$$
\begin{array}{lclcll}
0.3 \cdot 2 & = & 0.6 & \Rightarrow & \text{bit } 0 & [\text{peso } 0.5] \\
0.6 \cdot 2 & = & 1.2 & \Rightarrow & \text{bit } 1 & [\text{peso } 0.25] \\
0.2 \cdot 2 & = & 0.4 & \Rightarrow & \text{bit } 0 & [\text{peso } 0.125] \\
0.4 \cdot 2 & = & 0.8 & \Rightarrow & \text{bit } 0 & [\text{peso } 0.0625] \\
0.8 \cdot 2 & = & 1.6 & \Rightarrow & \text{bit } 1 & [\text{peso } 0.03125] \\
0.6 \cdot 2 & = & 1.2 & \Rightarrow & \text{bit } 1 & [\text{peso } 0.015625] \\
0.2 \cdot 2 & = & 0.4 & \Rightarrow & \text{bit } 0 & [\text{peso } 0.007815]
\end{array}
$$

Conversione frazionaria da 10 a 2



Conversione frazionaria da 2 a 10



Esempio

m=4	NFR	*i*
0.0101	1 / 2 = 0.5	-4
0.0101	(0.5 + 0)/2 = 0.25	-3
0.0101	(0.25 + 1)/2 =0.625	-2
0.0101	(0.625+0)/2= 0.3125	-1

$0.0101_{(2)} = 0.3125_{(10)}$

Numeri rappresentabili con n bit

- Avendo a disposizione n bit è possibile rappresentare 2^n combinazioni di cifre binarie, corrispondenti ai numeri da 0 a 2^{n-1}
- Per rappresentare i numeri da 0 a N-1 occorrono $\log 2^{(N)}$ bit (le parentesi hanno il significato di "il più piccolo numero intero maggiore o uguale al valore tra parentesi", nel nostro caso $\log 2^{(N)}$)
- Infatti per rappresentare le 8 cifre ottali servono
- 3 bit, mentre per rappresentare le 16 cifre esadecimali ne servono 4, se dovessimo rappresentare 12 cifre ne servirebbero ancora 4.

Trasformazione frazionaria da 2 a 8/16 e viceversa

- La procedura è la stessa di quella vista per la conversione della parte intera

``Si raggruppano le cifre a 3 a 3 o a 4 a 4 partendo dal . decimale e muovendosi verso sinistra per la parte intera e verso destra per la parte frazionaria''

$$\overbrace{1010}\,\overbrace{0010}\,\overbrace{1110}\,.\,\overbrace{1011}\,\overbrace{0101}\;_{(2)} \qquad A2E.B5\;_{(16)}$$

$$\overbrace{011}\,\overbrace{010}\,\overbrace{111}\,.\,\overbrace{011}\,\overbrace{101}\;_{(2)} \qquad 327.35\;_{(8)}$$

Aritmetica binaria: somma numeri naturali

- Le regole non sono diverse da quelle che conosciamo per la base 10: sommando numeri su più cifre si potrà avere un riporto (carry) mentre se si effettua una sottrazione potrebbe servire un prestito.

- **Tabella della somma:**
  - 0 + 0 = 0
  - 0 + 1 = 1
  - 1 + 0 = 1
  - 1 + 1 = 0 e riporto di 1

$$\begin{array}{r} 0\ + \\ 0\ = \\ \hline 0 \end{array} \qquad \begin{array}{r} 0\ + \\ 1\ = \\ \hline 1 \end{array} \qquad \begin{array}{r} 1\ + \\ 0\ = \\ \hline 1 \end{array} \qquad \begin{array}{r} 1\ + \\ 1\ = \\ \hline 1\ 0 \end{array}$$

- Esempio: 1010+111

```
 11
 1010+
  111
-----
10001
```

Aritmetica binaria: algoritmo di somma numeri naturali

- Si usa l'algoritmo chiamato *"addizione a propagazione del riporto";* esso è l'algoritmo elementare decimale, adattato però alla base 2. Esempio per n=8:

$$
\begin{array}{cccccccccc}
c_8 & c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & 0 & \\
 & x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 & + \\
 & y_7 & y_6 & y_5 & y_4 & y_3 & y_2 & y_1 & y_0 & = \\
\hline
 & s_7 & s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0 &
\end{array}
$$

Aritmetica binaria: algoritmo di somma numeri naturali

- Esempio:

Pesi	7	6	5	4	3	2	1	0		
Riporto			1	1	1					
Addendo 1	0	1	0	0	1	1	0	1	+	77_{dec}
Addendo 2	1	0	0	1	1	1	0	0	=	156_{dec}
Somma	1	1	1	0	1	0	0	1		233_{dec}

Aritmetica binaria: algoritmo di somma numeri naturali

- Addizione con riporto in uscita (*carry out*)

riporto in uscita

Pesi		7	6	5	4	3	2	1	0		
Riporto	1	1	1	1	1	1					
Addendo 1		0	1	1	1	1	1	0	1	+	125_{dec}
Addendo 2		1	0	0	1	1	1	0	0	=	156_{dec}
Somma		0	0	0	1	1	0	0	1		25_{dec} !

risultato errato!

Overflow

Aritmetica binaria: sottrazione numeri naturali

- **Tabella della sottrazione:**
  - 0 - 0 = 0
  - 0 - 1 = 1 e prestito di 1
  - 1 - 0 = 1
  - 1 - 1 = 0

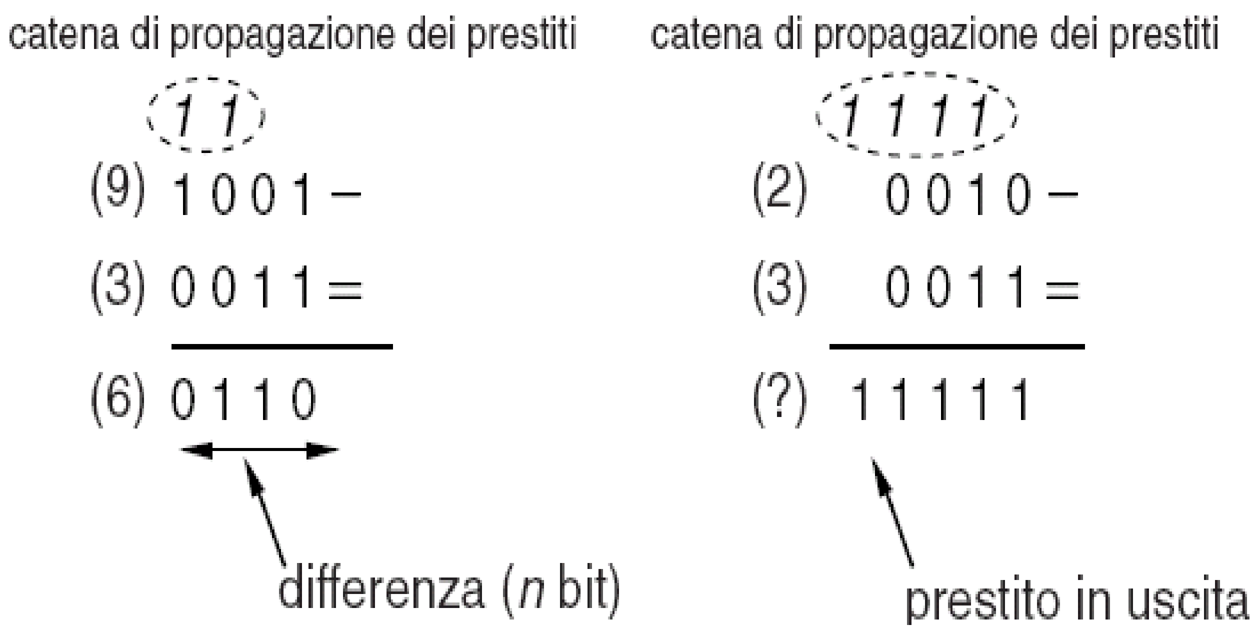
0 −	0 −	1 −	1 −
0 =	1 =	0 =	1 =
0	1 1	1	0

- Esempio: 1010-111

```
  11
 1010-
  111
-----
 0011
```

Aritmetica binaria: algoritmo di sottrazione dei numeri naturali

- L’algoritmo è quello manuale, con propagazione dei prestiti: *la sottrazione naturale è possibile solo se il minuendo è maggiore del o uguale al sottraendo*.



Aritmetica binaria: moltiplicazione numeri naturali

- **Tabella della moltiplicazione:**
  - 0 x 0 = 0
  - 0 x 1 = 0
  - 1 x 0 = 0
  - 1 x 1 = 1

$$\begin{array}{r} 0 \times \\ 0 = \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \times \\ 1 = \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \times \\ 0 = \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \times \\ 1 = \\ \hline 1 \end{array}$$

- Esempio: 1010x111

```
    1010x
     111
    -----
    1010+
   1010 +
  1010
 --------
 1000110
```

Aritmetica binaria: algoritmo di moltiplicazione dei numeri naturali

$$
\begin{array}{ccccccccc}
 & & & & x_3 & x_2 & x_1 & x_0 & \times \\
 & & & & y_3 & y_2 & y_1 & y_0 & = \\
\hline
 & & & & x_3y_0 & x_2y_0 & x_1y_0 & x_0y_0 & (P_0) \\
 & & & x_3y_1 & x_2y_1 & x_1y_1 & x_0y_1 & & (P_1) \\
 & & x_3y_2 & x_2y_2 & x_1y_2 & x_0y_2 & & & (P_2) \\
 & x_3y_3 & x_2y_3 & x_1y_3 & x_0y_3 & & & & (P_3) \\
\hline
q_7 & q_6 & q_5 & q_4 & q_3 & q_2 & q_1 & q_0 & (Q)
\end{array}
$$

- Si noti che: il risultato del prodotto di due numeri a 4 bit richiede 8 bit

$$(2^n - 1)(2^n - 1) = 2^{2n} - 2^n - 2^n + 1 > 2^{2n-1}$$

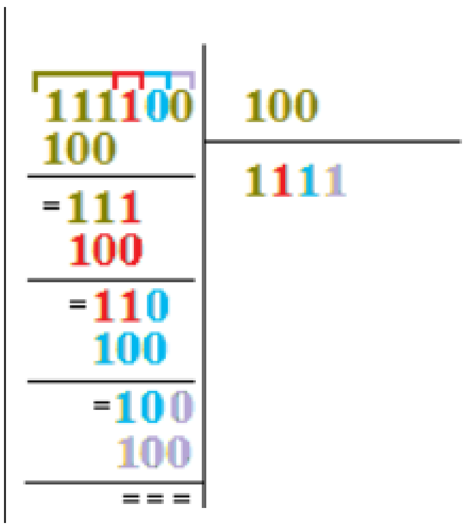
Aritmetica binaria: algoritmo di moltiplicazione dei numeri naturali

- Esempio

$$
\begin{array}{cccccccc|c}
 & & & & 1 & 0 & 1 & 1 & \times \\
 & & & & 0 & 1 & 1 & 0 & = \\
\cline{5-8}
 & & & & 0 & 0 & 0 & 0 & (P_0) \\
 & & & 1 & 0 & 1 & 1 & & (P_1) \\
 & & 1 & 0 & 1 & 1 & & & (P_2) \\
 & 0 & 0 & 0 & 0 & & & & (P_3) \\
\hline
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 &
\end{array}
$$

Aritmetica binaria: divisione numeri naturali

- **Tabella della divisione:**
  - 0 : 1 = 0
  - 1 : 1 = 1



- Esempio: 1100:11

```
1100:11
11   |1
 00
 11 |0
 00
 11 |0
```

- Esempio: 10101:11

```
10101:11
 11   |1
 100
  11 |1
  011
   11|1
```

Somma, Sottrazione, Moltiplicazione, Divisione in base 8 e 16

- Le regole sono le stesse

Esempio:

1	1	1	1		
	D	F	A	A	+
	B	B	B	B	
1	9	B	6	5	
	(1+13+11)-16	(1+15+11)-16	(1+10+11)-16	(10+11)-16	

		1	1	
D	F	A	A	−
B	B	B	B	
2	3	E	F	
13-11	(15-1)-11	16+(10-1)-11	16+10-11	

Numeri relativi (): rappresentazioni

Esistono tre rappresentazioni principali:

- **Modulo e segno (MS)**
- **Rappresentazione in complemento a 1 C1**
- **Rappresentazione in complemento a 2 C2**

Ci focalizzeremo principalmente su soltanto su M&S e C_2

Rappresentazione modulo e segno

- La rappresentazione in Modulo e Segno (M&S) su n cifre binarie utilizza 1 bit per il segno e n-1 bit per il modulo
- Si possono rappresentare i valori nel range:

$$-(2^{n-1}+1) < B < 2^{n-1}-1$$

Numeri negativi	$(B < 0)$:	da -0 a $-(2^{n-1}+1)$
Numeri positivi	$(B > 0)$:	da 0 a $2^{n-1}-1$

- NOTA: Lo zero è rappresentato 2 volte
  - (con segno + e con segno -)

Rappresentazione modulo e segno

- ***Numeri binari interi*** (positivi e negativi) ***in MS***:
  - il primo bit (quello a sinistra) rappresenta il segno del numero,
    - 0 per segno positivo
    - 1 per segno negativo
  - mentre i bit rimanenti ne rappresentano il valore assoluto

- Esempio per una parola di n=4 bit:

$$5_{10} \rightarrow 0101_2$$
$$-5_{10} \rightarrow 1101_2$$

Rappresentazione modulo e segno

Osservazioni:

- Il bit di segno è *applicato* al numero rappresentato, ma non fa propriamente *parte* del numero in quanto tale, perché il bit di segno non ha significato numerico.

- *Distaccando il bit di segno*, i bit rimanenti rappresentano il valore assoluto del numero.

Conversione da base 10 con segno alla rappresentazione M&S su N bit

- Nella conversion risulta importante definire su quanti bit si intende convertire il numero:

  $+10_{(MS)}$ su 5 bit 01010 , su 7 0001010

  $-7_{(MS)}$ su 5 bit 10111 , su 7 1000111

- Verificare che sia rappresentabile su N bit, quindi
  - Numero positivo: procedere come nel caso senza segno ed aggiungere eventuali 0 a sinistra per arrivare a N bit.
  - Numero negativo: ricavare la rappresentazione del corrispondente numero positivo e poi sostituire il bit di segno con 1

Conversione da base 10 con segno alla rappresentazione M&S su N bit

Convertire i seguenti numeri con segno in base 10 su 5 bit:

N1 = +12; N2 = +17; N3= -4; N4= -15; N5 = -16

- N1: $12_{10} = 1100_2 \rightarrow 01100_{(MS)}$
- N2: +17 non si può rappresentare in M&S su 5 bit
- N3: $4_{10} = 100_2 \rightarrow 00100 \rightarrow -4_{10} = 10100_{M\&S}$ su 5 bit
- N4: $15_{10} = 01111_2 \rightarrow -15_{10} = 11111_{M\&S}$
- N5: -16 NON si può rappresentare in M&S su 5 bit

Operazioni in MS

- Il calcolo di somme e sottrazioni in MS richiede di analizzare segno e modulo degli operandi per individuare il segno del risultato e l'operazione da eseguire sui moduli per ottenere il modulo del risultato.

Somma in MS

Segni concordi

- Il segno del risultato è uguale al segno degli operandi
- Il modulo del risultato si ottiene sommando i moduli degli operandi

Segni discordi

- Il segno del risultato è uguale al segno dell'operando con modulo più grande
- Il modulo del risultato si ottiene sottraendo il modulo più piccolo dal modulo più grande

- **In quale caso si potrebbe verificare OVERFLOW?**

Sottrazione in MS

Segni concordi

- Il segno del risultato dipende sia dal segno che dal modulo di minuendo (n_1) e sottraendo (n_2):
  - Se sono entrambi positivi il segno è positivo se $|n1| \geq |n2|$, negativo altrimenti
  - Se sono entrambi negativi il segno è negativo se $|n1| > |n2|$, positivo altrimenti
- Il modulo del risultato si ottiene sottraendo il modulo più piccolo dal modulo più grande

Sottrazione in MS

Segni discordi

- Il segno del risultato è uguale al segno del minuendo
- Il modulo del risultato si ottiene sommando i due moduli

In quale caso si potrebbe verificare OVERFLOW?

Moltiplicazione in MS

Segno di X	Segno di Y	x_{n-1}	y_{n-1}	Segno di Q	q_{2n-2}
+	+	0	0	+	0
+	−	0	1	−	1
−	+	1	0	−	1
−	−	1	1	+	0

Complemento a 1

- Sia n il numero di bit di una parola:

$$-(2^{n-1}+1) < B < 2^{n-1}-1$$

Numeri negativi	(B < 0):	da -0 a $-(2^{n-1}+1)$
Numeri positivi	(B>0):	da 0 a $2^{n-1}-1$

- Il cambiamento di segno si ottiene complementando ciascun bit.
- Esempio
- per n=4 bit: $5_{10} \rightarrow 0101_{(C1)}$

  $-5_{10} \rightarrow 1010_{(C1)}$

Complemento a 2

- Sia n il numero di bit di una parola:

$$-(2^{n-1}) < B < 2^{n-1}-1$$

Numeri negativi	(B < 0):	da -1 a -2^{n-1}
Numeri positivi	(B>0):	da 0 a $2^{n-1}-1$

- Il cambiamento di segno si ottiene complimentando a 1 e quindi sommando 1 al numero ottenuto.

- Esempio per n=4 bit:
- $5_{10} \rightarrow 0101_{(C2)}$
- $-5_{10} \rightarrow 1010 + 0001 = 1011_{(C2)}$

Complemento a 2

Esempo: numero a 3 bit in complemento a 2

- $000_{C2} = -0\times2^2 + 0\times2^1 + 0\times2^0 = 0_{10}$
- $001_{C2} = -0\times2^2 + 0\times2^1 + 1\times2^0 = 1_{10}$
- $010_{C2} = -0\times2^2 + 1\times2^1 + 0\times2^0 = 2_{10}$
- $011_{C2} = -0\times2^2 + 1\times2^1 + 1\times2^0 = 2 + 1 = 3_{10}$
- $100_{C2} = -1\times2^2 + 0\times2^1 + 0\times2^0 = -4_{10}$
- $101_{C2} = -1\times2^2 + 0\times2^1 + 1\times2^0 = -4 + 1 = -3_{10}$
- $110_{C2} = -1\times2^2 + 1\times2^1 + 0\times2^0 = -4 + 2 = -2_{10}$
- $111_{C2} = -1\times2^2 + 1\times2^1 + 1\times2^0 = -4 + 2 + 1 = -1_{10}$

Generalizzazione degli operatori

- I due operatori si possono generalizzare a una base r qualsiasi: si indicano con i termini *complemento alla base* e *complemento alla base diminuita*

- L'operatore di complemento a 2 è il complemento alla base per la base 2, quello di complemento a 1 è il complemento alla base diminuita per la stessa base

- Se la base fosse 10, il complemento alla base sarebbe il complemento a 10 mentre il complemento alla base diminuita sarebbe il complemento a 9

Aumento dei Bit in C2

- Data la rappresentazione di un numero in C2 su k bit possiamo ottenere la rappresentazione dello stesso numero su n>k bit *estendendo il segno* cioè **replicando il bit di segno negli n-k bit più significativi**. Per esempio:

$$4 = 0100 = 00100 = 000100 = \ldots\text{(indefinitamente)}$$

$$-5 = 1011 = 11011 = 111011 = \ldots\text{(indefinitamente)}$$

Numero in base 10 con segno	Rappresentazione C2 su 4 bit	Rappresentazione C2 su 8 bit
+5	0101	00000101
-1	1111	11111111
-8	1000	11111000
+7	0111	00000111

Riduzione dei Bit in C2

- *La contrazione* del segno o riduzione di bit, avviene ***cancellando*** in modo progressivo **il bit di segno a sinistra**, il valore del numero non muta, *purché così facendo il bit di segno non abbia a invertirsi* ! Per esempio:
  - 7= 000111 = 00111 = 0111 STOP! (111 è < 0)
  - –3 = 111101 = 11101 = 1101 = 101 STOP! (01 è > 0)

Segno in C2: Osservazioni

- *Il segno è incorporato nel numero rappresentato in* C_2, non è semplicemente *applicato* (come in segno e valore assoluto).
- *Il bit più significativo rivela il segno*: 0 per numero positivo, 1 per numero negativo (il numero zero è considerato positivo), ma …
- <u>NON si può *distaccare* il bit più significativo e dire che i bit rimanenti rappresentino il puro valore, senza segno, del numero (ciò è vero solo se il numero è positivo) !</u>

Conversione da Decimale a C2

Algoritmo:

- Se $D_{dec} \geq 0$:
  - Verificare che D_{dec} è convertibile in converti n-1 bit
  - converti D_{10} in binario naturale
  - premetti 0 alla sequenza di bit ottenuta fino ad arrivare a n bit

esempio: $154_{10} \Rightarrow 10011010_2 \Rightarrow 010011010_{C2}$

Conversione da Decimale a C2

- Se $N_{10} < 0$:
  - Verificare che D_{dec} è convertibile in converti n-1 bit
  - trascura il segno e converti D_{10} in binario naturale
  - premetti 0 alla sequenza di bit ottenuta fino ad arrivare a n bit
  - calcola l'opposto del numero così ottenuto, secondo la procedura di inversione in C_2
  - esempio: $-154_{10} \Rightarrow 154_{10} \Rightarrow 10011010_2 \Rightarrow$
  - $\Rightarrow 010011010_2 \Rightarrow 101100101 + 1 \Rightarrow 101100110_{C2}$
- Occorrono 9 bit sia per 154_{10} sia per -154_{10}

Conversione da Decimale a C2: ESEMPI

Convertire i seguenti numeri con segno in base 10 su 5 bit:

N1 = +12; N2 = +17; N3=-8; N4=-16

- N1: $12_{10} = 1100_2 \rightarrow 01100_{C2}$ su 5 bit

- N2: il massimo numero positivo rappresentabile in C2 su 5 bit è $2^{5-1} - 1 = 16 - 1$ =+ 15, quindi +17 non si può rappresentare in C2 su 5 bit

- N3: $8_{10} = 1000_2 \rightarrow 01000$ applichiamo l'operatore complemento a 2

  01000 = 11000 quindi $-8_{10} = 11000_{C2}$ su 5 bit

- N4: il massimo numero negativo rappresentabile su 5 bit è -2^{5-1} ovvero -16 che si rappresenta con un 1 nel bit più significativo seguito da quattro 0 $-16_{10} = 10000_{C2}$

Conversione da Decimale a C2

- La conversione del seguente numero in C2

$$N_{C2} = b_{k-1} b_{n-2} \dots b_0$$

alla base 10 con segno si può effettuare in due modi:

  - in modo diretto tramite la formula

$$V(N_{C2}) = -2^{k-1} \cdot b_{k-1} + \sum_{i=0}^{k-2} 2^i \cdot b_i$$

  - distinguendo due casi:
    - Se il bit più significativo b_{k-1} è 0, il segno è + e il valore si calcola come visto per i numeri senza segno
    - Se il bit più significativo b_{k-1} è 1, il segno è – e il valore si calcola applicando l'algoritmo di trasformazione dei numeri senza segno al complemento a 2 del numero: N (bit di segno incluso, così vale anche per i casi particolari)

Conversione da Decimale a C2: Esempi

Convertire i seguenti numeri binari codificati secondo la rappresentazione C2 nelle corrispondenti rappresentazioni in base 10 con segno

N1 = 0111 N2 = 1011

Metodo 1

N1: $-0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = +7_{10}$

N2: $-1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = -5_{10}$

Metodo 2

Segno N1: +; Conversione in base 10 di 0111: 7 ➡ $+7_{10}$

Segno N2: -; Conversione in base 10 di 0101: 5 ➡ -5_{10}

Riepilogo rappresentazione interi su 4 bit

Dec.	Segno e modulo	Compl. a 1	Compl. a 2
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
0	0000	0000	0000

Dec.	Segno e modulo	Compl. a 1	Compl. a 2
-0	1000	1111	-
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	-	-	1000

Confronto tra Rappresentazioni

B				valore rappresentato			
b_3	b_2	b_1	b_0	**naturale**	**segno e val. ass.**	**comp. a 1**	**comp. a 2**
0	1	1	1	+7	+7	+7	+7
0	1	1	0	+6	+6	+6	+6
0	1	0	1	+5	+5	+5	+5
0	1	0	0	+4	+4	+4	+4
0	0	1	1	+3	+3	+3	+3
0	0	1	0	+2	+2	+2	+2
0	0	0	1	+1	+1	+1	+1
0	0	0	0	+0	+0	+0	+0
1	0	0	0	+8	−0	−7	−8
1	0	0	1	+9	−1	−6	−7
1	0	1	0	+10	−2	−5	−6
1	0	1	1	+11	−3	−4	−5
1	1	0	0	+12	−4	−3	−4
1	1	0	1	+13	−5	−2	−3
1	1	1	0	+14	−6	−1	−2
1	1	1	1	+15	−7	−0	−1

Addizioni e sottrazioni tra numeri in C2

- La somma e la sottrazione di numeri in complemento a 2 risulta più agevole rispetto a quella vista per la rappresentazione M&S:
  - la somma si esegue come per i numeri senza segno
  - la differenza $n_1 - n_2$ diventa una somma $n1 + \quad n_2$

in entrambi i casi si scarta l'eventuale riporto *oltre* la cifra del segno

- La somma di numeri concordi o la differenza di numeri discordi possono portare ad overflow (come nel caso della rappresentazione in M&S

Esempi di addizione in complemento a 2

- L'algoritmo è *identico* a quello naturale! (come se il bit di segno non fosse negativo)

Pesi	7	6	5	4	3	2	1	0		
Riporto			1	1	1					
Addendo 1	0	1	0	0	1	1	0	1	+	77_{dec}
Addendo 2	1	0	0	1	1	1	0	0	=	-100_{dec}
Somma	1	1	1	0	1	0	0	1		-23_{dec}

Esempi di addizione in complemento a 2

- -2 + 3 (in complemento a 2 su 3 bit)

Riporto 1	1			
-2	1	1	0	+
+3	0	1	1	=
-2 + 3 = +1	0	0	1	

Sommando le rappresentazioni in C2 di -2 e + 3 si ottiene correttamente la rappresentazione di +1. Il riporto generato sulla cifra più significativa si può semplicemente scartare.

Esempi di addizione in complemento a 2

- -3 + (-1)

Riporto 1	1	1		
-3	1	0	1	+
-1	1	1	1	=
-3 + (-1) = -4	1	0	0	

- -2 + (-3)

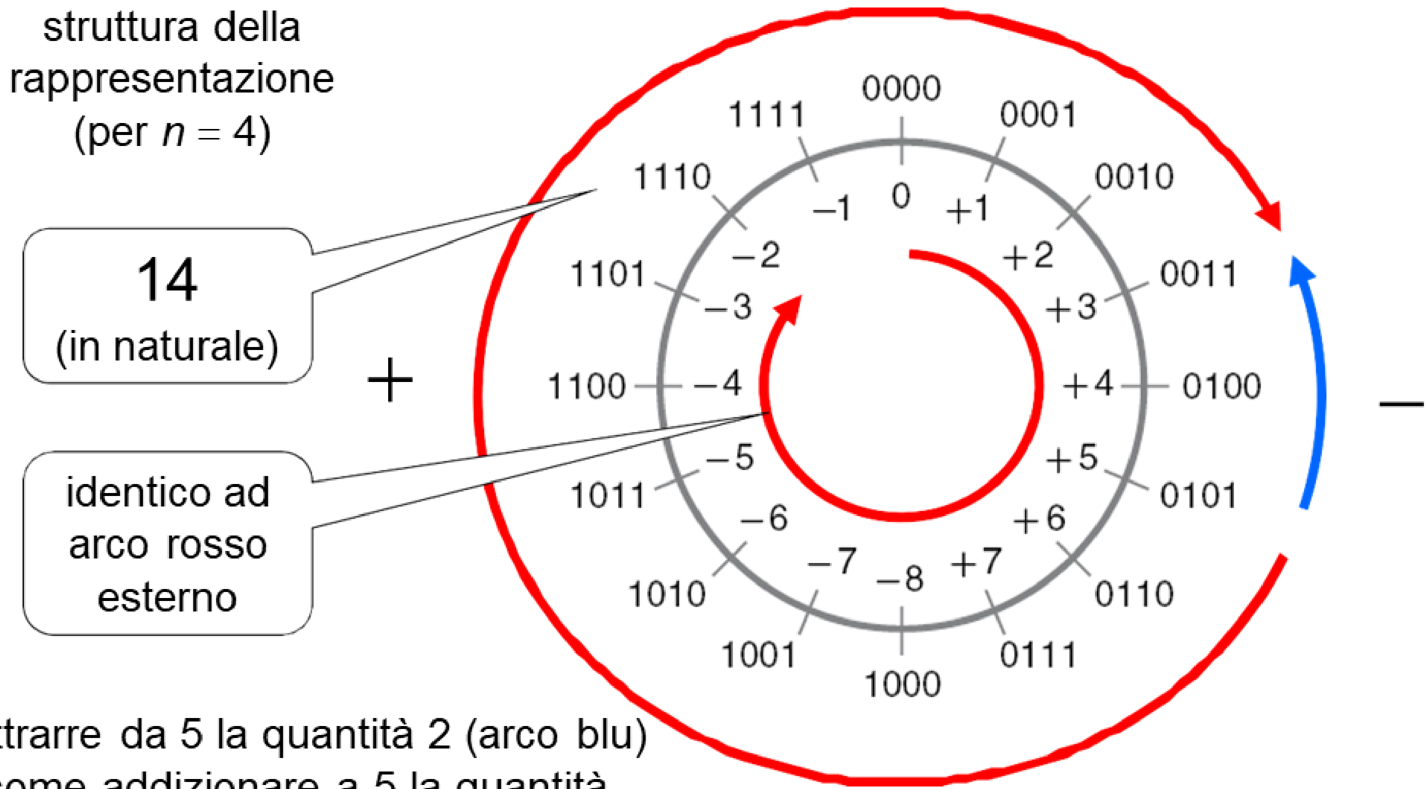
Riporto 1				
-2	1	1	0	+
-3	1	0	1	=
-2 + (-3) = !!!	0	0	1	

Si è verificato OVERFLOW: il segno del risultato è diverso da quello degli operandi (che avevano segni *concordi*).

Esempi di addizione in complemento a 2

0 0 1 0 +	(+2)		(b)	0 1 0 0 +	(+4)
0 0 1 1 =	(+3)			1 0 1 0 =	(−6)
0 1 0 1	(+5)			1 1 1 0	(−2)
1 0 1 1 +	(−5)		(d)	0 1 1 1 +	(+7)
1 1 1 0 =	(−2)			1 1 0 1 =	(−3)
1 0 0 1	(7)			0 1 0 0	(+4)
1 1 0 1 −	(−3)			1 1 0 1 +	
1 0 0 1 =	(−7)	⇒		0 1 1 1 =	
				0 1 0 0	(+4)
0 0 1 0 −	(+2)			0 0 1 0 +	
0 1 0 0 =	(+4)	⇒		1 1 0 0 =	
				1 1 1 0	(−2)

Come funziona l'addizione algebrica in C2



sottrarre da 5 la quantità 2 (arco blu)
è come addizionare a 5 la quantità
16 − 2 = 14 (arco rosso esterno),
trascurando il riporto in uscita

Come rilevare l'overflow in C2

- *Quando gli addendi sono discordi* (cioè di segno diverso) *non si verifica mai trabocco*:
  - il valore assoluto della differenza è sempre minore del valore assoluto del minuendo o del sottraendo !

- *Quando gli addendi sono concordi* (di segno uguale), *si verifica trabocco se e solo se il segno del risultato non concorda con quello degli addendi*, cioè se si hanno:
  - addendi positivi ma risultato negativo
  - addendi negativi ma risultato positivo

Sottrazione in C2

- Per sottrarre in complemento a due, basta prima invertire (in C2) il sottraendo, e poi addizionare.
- Salvo il verificarsi di trabocco, la sottrazione in C2 non ha restrizioni.
- In effetti, in C2 addizione e sottrazione si possono riguardare essenzialmente come la medesima operazione: addizione algebrica.
- È uno dei motivi (forse il principale) che fanno del complemento a due la tecnica di rappresentazione preferita nel calcolatore.

Esempi sottrazione usando il C2

- n=8, 30 - 22
  - $30_{10} \rightarrow 00011110_2$
  - $22_{10} \rightarrow 00010110_2$, trasformo 22 in compl. a 2:
  - $-22_{10} \rightarrow 11101001_2+1 = 11101010_2$
  - Ora sommo a 30 il -22 così ottenuto:
    - $30_{10} \rightarrow 00011110_2+$
    - $-22_{10} \rightarrow 11101010_2=$
    - ---------
    - *1*00001000
    - ↑ c'è riporto! → **il risultato è positivo**.

Esempi sottrazione usando il C2

- n=8, 19 - 22
  - $19_{10} \rightarrow 00010011_2$
  - $22_{10} \rightarrow 00010110_2$, trasformo 22 in compl. a 2:
  - $-22_{10} \rightarrow 11101001_2+1 = 11101010_2$
  - Ora sommo a 19 il -22 così ottenuto:

$$
\begin{array}{rcl}
19_{10} & \rightarrow & 00010011_2 + \\
-22_{10} & \rightarrow & 11101010_2 = \\
 & & \text{---------} \\
 & & \_11111101 = -3_{10}
\end{array}
$$

↑ non c'è riporto! → **Il risultato è negativo.**

Osservazioni sulla rappresentazione in C2

- La tecnica di rappresentazione in complemento a due oggi è quella più usata nel calcolatore.
- Rispetto alle altre due (naturale e segno-valore assoluto), è un buon compromesso tra:
  - semplicità di definizione
  - capacità di rappresentare i numeri
  - ed efficienza delle operazioni
- Esistono algoritmi (e circuiti digitali) per calcolare anche moltiplicazione, divisione (intera), resto e quoziente, ecc.
- Le stesse operazioni di addizione e sottrazione ammettono svariati altri algoritmi (oltre a quello a propagazione di riporto o prestito), talvolta molto raffinati ed efficienti.

Codici

- In un sistema di numerazione in base r, con n cifre è possibile rappresentare:
  - *r^n numeri interi, da 0… $r^n - 1$*
  - *I numeri frazionari da 0.0 … $(r^n - 1)/ r^n$*
- Un codice binario a n-bit è un gruppo di n bit, che può assumere fino a 2^n combinazioni distinte di 1 e 0, in cui ciascuna combinazione rappresenta un elemento dell'insieme da codificare.
  - *Un insieme di quattro elementi può essere codificato con un codice binario con n = 2 bit (00, 01, 10, 11)*
  - *Un insieme di otto elementi richiederà un codice a 3 bit*
  - *Un insieme di 16 elementi richiederà un codice a 4 bit*

BCD (Binary Coded Decimal)

- Affinché il codice sia non ambiguo ad ogni elemento dell'insieme C deve corrispondere una ed una sola combinazione di bit.
- Se l'insieme da codificare è quello dei simboli del sistema decimale sono necessari n = 4 bit.
- Sono una classe di codici che permette di rappresentare le cifre decimali con simboli binari
- Un codice si dice pesato se si può attribuire un peso alla presenza del simbolo 1 in una determinata posizione.

- BCD8421: pesi 8,4,2,1.

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

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

Codice BCD

- Un numero con n simboli decimali richiederà 4xn bit in BCD
- $12_{10} = (0001\ 0010)_{BCD}$
- 2 simboli decimali => 4x2 = 8 bit
- $(12)_{10}=(1100)_2$ ma $1100 \notin$ BCD

- Con 4 bit sono possibili 16 combinazioni distinte
- In BCD 8421, le sei combinazioni seguenti non fanno parte del codice
  - *1010* (10_{10})
  - *1011* (11_{10})
  - *1100* (12_{10})
  - *1101* (13_{10})
  - *1110* (14_{10})
  - *1111* (15_{10})

Somma in BCD

- La somma di due cifre decimali con l'eventuale riporto varia tra 0 … (9+9+1)=19 ovvero tra $(00000)_2$ e $(10011)_2$
- In BCD la somma deve variare tra $(0\ 0000)_{BCD}$ e $(1\ 1001)_{BCD}$
- Per ottenere la cifra BCD corretta, quando la somma binaria supera 1001_2 occorre sommare $6_{10} = 0110_2$ alla somma ottenuta e aggiungere un riporto alla colonna immediatamente più significativa
- Sommare in BCD $448_{10} + 489_{10}$

110	Riporto BCD	1	1	
448		0100	0100	1000
+489		+0100	+1000	+1001
937	Somma binaria	1001	1101	1 0001
	Somma di 6		+0110	+0110
	Somma BCD		1 0011	1 0111
	Risultato BCD	1001	0011	0111

Numeri razionali

- Rappresentazione in virgola fissa

- Rappresentazione in virgola mobile e standard IEEE-754

Numeri frazionari in virgola fissa

- La rappresentazione in virgola fissa si basa sull'idea di suddividere la sequenza di bit rappresentante il numero frazionario in due parti di lunghezza prefissata (ma non necessariamente uguale): parte intera e frazionaria.

- Il numero di bit a sinistra e destra della virgola è stabilito a priori, anche se alcuni bit restano nulli.

Numeri frazionari in virgola fissa

- Dato che la posizione delle virgola è stabilita a priori, addizione e sottrazione si svolgono come con i numeri interi (gli algoritmi sono sostanzialmente gli stessi).

- La virgola fissa è una rappresentazione semplice, ma poco flessibile, e può condurre a spreco di bit.

- Inoltre, per rappresentare numeri grandi oppure precisi occorrono molti bit.

Numeri in virgola mobile

- Si fa uso di rappresentazioni normalizzate aventi lo scopo di non doversi curare della posizione della virgola e del numero delle cifre utilizzate.
- Queste rappresentazioni vengono espresse come il prodotto di due fattori:
  - le cifre significative del numero,
  - una potenza del 10 il cui esponente definisce la posizione della virgola nel numero.
- Si usa spesso la rappresentazione in virgola mobile (floating point) anche in base 10, dove è chiamata notazione scientifica:

$$0{,}137 \times 10^{8}{}_{\text{notazione scientifica}} \quad \text{vale} \quad 13.700.000_{\text{dec}}$$

Notazione scientifica

$$N = f \times b^e$$

dove

f è la frazione o mantissa

b è la base

e è l'esponente (un intero positivo o negativo)

Esempi:

$$\begin{aligned} 3.14 &= 0.314 \times 10^1 = 31.4 \times 10^{-1} \\ 0.000001 &= 0.01 \times 10^{-4} = 1.0 \times 10^{-6} \\ 1941 &= 0.1941 \times 10^4 = 19.41 \times 10^2 \end{aligned}$$

Rappresentazione esponenziale normalizzata

- Per rendere la rappresentazione in virgola mobile univoca si definisce la rappresentazione **normalizzata**

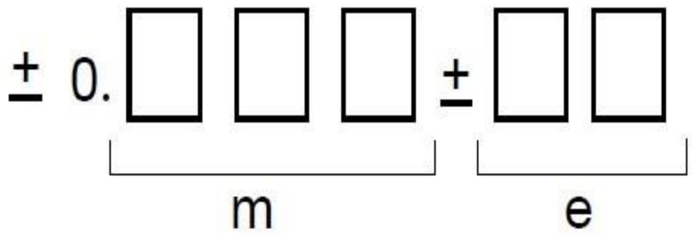
Convenzione: la prima cifra significativa si trova immediatamente a destra della virgola.

$$0.XXXXX$$
$$\uparrow$$

- A tal fine basta aumentare o diminuire il valore dell'esponente di tante unità quante sono le posizioni di cui è stata spostata la virgola.
  - Esempi: $127x10^{6} = 0{,}127x10^{9}$
    - $15x10^{-7} = 0{,}15x10^{-5}$

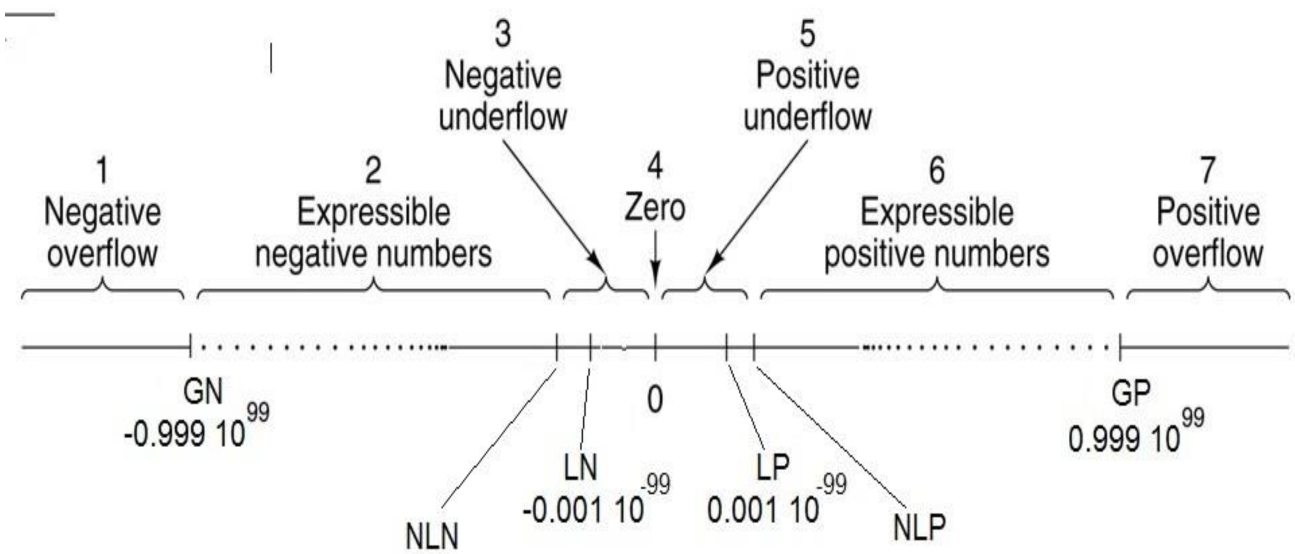
Range di valori rappresentabili in virgola mobile

- Esempio di numero in virgola mobile in base 10. La parte frazionaria ha 3 cifre mentre l'esponente ne ha 2



Range di valori rappresentabili in virgola mobile

- *NGP* - Normalized Greatest Positive = $+0.999 \times 10^{99}$
- *NGN* - Normalized Greatest Negative = -0.999×10^{99}
- *NLP* - Normalized Lowest Positive = $+0.100 \times 10^{-99}$
- *NLN* - Normalized Lowest Negative = -0.100×10^{-99}



Principi base della rappresentazione in virgola mobile

- L'insieme dei numeri in virgola mobile non rappresenta un insieme continuo:

$$\pm\, 0.\underbrace{\square\,\square\,\square}_{m}\ \underbrace{\pm\,\square\,\square}_{e}$$

  - Da m = 0.100 a m = 0.999 ci sono 900 valori frazionari distinti.
  - Da e = −99 a e = +99 ci sono 199 ordini di grandezza.
- Quindi la rappresentazione comprende:
  - 900 ∗ 199= **179100**   **numeri positivi**
  - 900 ∗ 199= **179100**   **numeri negativi**
  - 1 zero

Totale :358201 numeri in virgola mobile (Floating Point – FlP)

Granularità dei numeri in virgola mobile

La granularità della rappresentazione dipende dall'esponente:

$$\pm\ 0.\underbrace{\square\,\square\,\square}_{m}\ \pm\underbrace{\square\,\square}_{e}$$

Fissato un esponente e la granularità è

$$g = 0.001 \times 10^{e}$$

Pertanto i punti sono più densi in corrispondenza degli esponenti più bassi.

$$\begin{array}{lll}
e = +99 & \quad g & = 0.001 \times 10^{+99} \\
e = +00 & \quad g & = 0.001 \times 10^{+00} \\
e = -99 & \quad g & = 0.001 \times 10^{-99}
\end{array}$$

Approssimazione nei numeri in virgola mobile

Esempio di programma per verificare l'approssimazione nei calcoli con numeri in virgola mobile.
Dato un numero reale k, iniziare con

$$u_0 = 1/k$$

Quindi calcolare i seguenti numeri definiti in modo ricorsivo per $i = 1, \ldots, n$

$$u_i = (1 + k) * u_{i-1} - 1$$

Notare che tutti i valori ottenuti sono sempre uguali a $1/k$, infatti

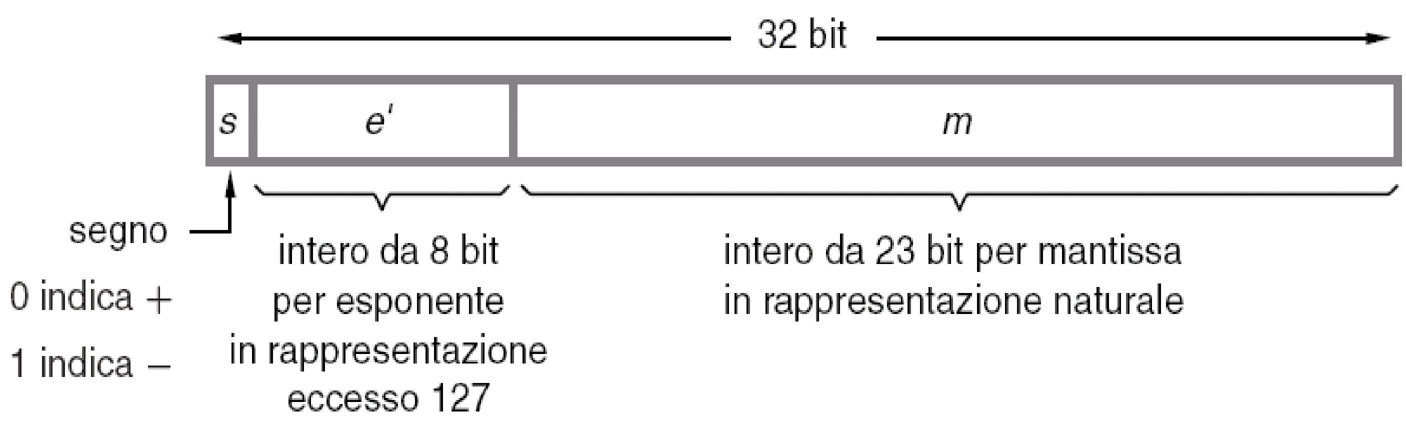
$$u_1 = (1 + k) * u_0 - 1 = (1 + k)/k - 1 = 1/k$$

IEEE 754: Standard Binary Floating-Point Arithmetic

- **Lo standard IEEE 754** regola l’aritmetica dei numeri binari in virgola mobile. Esso stabilisce il formato di rappresentazione dei numeri, le operazioni di base, le conversioni e le condizioni eccezionali.
- Lo standard definisce diversi formati
  - Precisione singola (32 bits)
  - Precisione doppia (64 bits)
  - Precisione quadrupla (128 bits)

IEEE 754: Standard Binary Floating-Point Arithmetic

- La rappresentazione dei numeri in virgola mobile con precisione singola ha il seguente formato:
  - 1 bit rappresenta il segno s
  - 8 bit rappresentano l'esponente e
  - 23 bit rappresentano la parte frazionaria (significando) m



IEEE 754: Il significando

Lo standard IEEE 754 definisce la parte frazionaria in un modo particolare.

Consiste di un bit 1 *implicito* prima della virgola, e una virgola implicita di seguito all'1 implicito, e viene chiamata *significando*.

Poichè il bit più significativo è sempre uguale a 1 per un numero normalizato questo numero non viene rappresentato e viene chiamato *bit nascosto (hidden bit)*.

$$\underbrace{1\,.}_{\textit{hidden bit}}\quad \underbrace{xxxxxxxx\ldots x}_{\text{23 } \textit{bits}}$$

Vengono memorizzati solo i 23 bit che seguono la virgola implicita.

IEEE 754: L'esponente

Lo standard IEEE 754 definisce l'esponente e come un numero di 8 bit nella forma *eccesso* $P = 127$

In questa rappresentazione un valore binario V è rappresentato dal valore

$$R = V + P$$

dove $P = 127$ è la costante di polarizzazione o eccesso.

ESEMPIO

Un esponente $V = -3$ è rappresentato dal numero binario
$R = -3 + 127 = 124$

Un esponente $V = -3$ è rappresentato dal numero binario
$R = +3 + 127 = 130$

IEEE 754: L'esponente

Rappresentazione dell'esponente R		*Vero esponente V*
binario	*decimale*	*decimale*
00000000	0	xx
00000001	1	-126
00000010	2	-125
00000011	3	-124
...	...	...
01111101	125	-2
01111110	126	-1
01111111	127	0
10000000	128	+1
10000001	129	+2
10000010	125	+3
...	...	...
11111100	252	+125
11111101	253	+126
11111110	254	+127
11111111	255	xx

L'esponente spazia nel seguente intervallo di valori:

−126 a +127

I due valori estremi sono

- R = 00000000 che corrisponde a V = −127
- R = 11111111 che corresponde a V = +128

sono utilizzati per un uso speciale.

Conversione decimale a IEEE 754: Esempio

$N = -13.25_{10}$

$\downarrow$ Convertire il valore assoluto del numero in binario

1101.01

$\downarrow$ Trasformare in forma normalizzata con bit nascosto

$1.\underbrace{10101}_{fraction} \times 2^3$

$\downarrow$ Calcolare l'esponente su 8 bit secondo la rappresentazione eccesso 127

$3 + 127 = 130 \longrightarrow 10000010$

$\downarrow$ Segno negativo

1

$\downarrow$ Rappresentazione su 32 bit

$$\underbrace{1}_{segno}\ \underbrace{10000010}_{esponente}\ \underbrace{10101000\cdots}_{frazione\ 23\ bits}$$

Conversione IEEE 754 a decimale

Un numero nel formato IEEE 754 è nella forma (s, e, f), dove

- $s \rightarrow$ 1 bit di segno
- $e \rightarrow$ 8 bit di esponente ($0 < e < 255$)
- $f \rightarrow$ 23 bit di parte frazionaria con bit nascosto

- Il numero normalizzato è nella forma:

$$N = (-1)^s \times 2^{(e-127)} \times (1.f)$$

Conversione IEEE 754 a decimale: Esempio

$N = BEC80000_{16}$

$N = 1011\ 1110\ 1100\ 1000\ 0000\ 0000\ 0000\ 0000$

$\downarrow$ partizione dei campi

$$\underbrace{1}_{sign}\ \underbrace{01111101}_{exponent}\ \underbrace{10010000000000000000000}_{23\ bits\ fraction}$$

$\downarrow$ Segno negativo

1

$\downarrow$ Esponente

$01111101 = 125_{10} \longrightarrow V = 125 - 127 = -2$

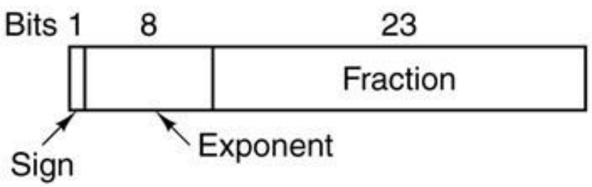
$\downarrow$ parte frazionaria con l'aggiunta del bit nascosto.

$1.1001000\cdots$

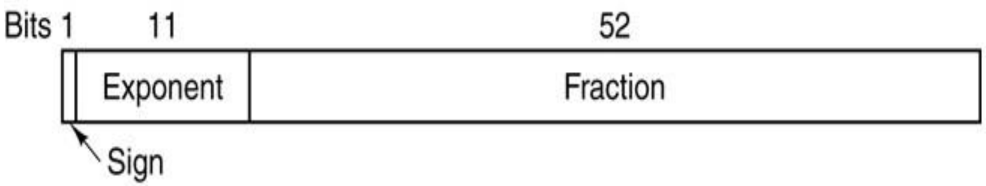
$\downarrow$ Conversione finale

$$N_{(2)} = -1.1001 \times 2^{-2} = -0.011001 = -0.390625_{(10)}$$

I formati dello standard IEEE 754



(a) Single precision



(b) Double precision

Item	Single precision	Double precision
Bits in sign	1	1
Bits in exponent	8	11
Bits in fraction	23	52
Bits, total	32	64
Exponent system	Excess 127	Excess 1023
Exponent range	−126 to +127	−1022 to +1023
Smallest normalized number	2^{-126}	2^{-1022}
Largest normalized number	approx. 2^{128}	approx. 2^{1024}
Decimal range	approx. 10^{-38} to 10^{38}	approx. 10^{-308} to 10^{308}
Smallest denormalized number	approx. 10^{-45}	approx. 10^{-324}

Osservazioni sulla rappresentazione floating-point

- Il formato di virgola mobile permette di rappresentare efficacemente sia numeri estremamente grandi sia numeri vicinissimi a zero, con molte cifre frazionarie.
- In altre parole, esso combina estensione e precisione.
- Per questo motivo tale formato è di solito quello preferito nel calcolatore per rappresentare il numero frazionario.
- Ciò vale specialmente nei calcoli di tipo scientifico, dove spesso occorre avere sia estensione sia precisione.

Aritmetica dei numeri floating-point

- **Moltiplicazione** - Gli esponenti vengono sommati e le mantisse moltiplicate. Alla fine il numero viene rinormalizzato se necessario.
- **Divisione** - Gli esponenti vengono sottratti e le frazioni divise. Alla fine il numero viene rinormalizzato se necessario.
- **Addizione e Sottrazione-** i numeri devono essere trasformati in una rappresentazione con lo stesso esponente (cosa che può far perdere in precisione) dopo di che le frazioni possono essere sommate. Alla fine il numero viene rinormalizzato se necessario.