
Calcolatori Elettronici

Rappresentazione dell'informazione

Ing. Gestionale e delle Telecomunicazioni
A.A. 2008/09
Gabriele Cecchetti

Rappresentazione dell'informazione

- **Sommario:**
 - Numerazione posizionale
 - Conversione tra basi diverse
 - Aritmetica binaria
 - Numeri relativi
 - Numeri frazionari
 - Numeri in virgola mobile
- **Riferimenti:**
 - Hamacher "Introduzione all'architettura del Calcolatore", cap. 3, sez. 3.1, 3.2, 3.3, 3.4 e 3.5
 - G. Bucci "Architetture e organizzazione dei Calcolatori Elettronici – Fondamenti", Cap. 2, tutte le sezioni eccetto 2.6.2 e 2.6.3

Cos'è un numero ? - Notazione posizionale

- Il **NUMERO** è un concetto astratto e corrisponde alla descrizione quantitativa degli oggetti contenuti in un dato insieme.
- Un **SISTEMA DI NUMERAZIONE** è un insieme di simboli e regole atti a rappresentare numeri.
- **Notazione posizionale**: permette di rappresentare un numero naturale qualsiasi (intero non negativo).

Sistema posizionale decimale

- La notazione decimale tradizionale è di tipo posizionale; esempio:

$$2718 = 2 \times 10^3 + 7 \times 10^2 + 1 \times 10^1 + 8 \times 10^0$$

Sistema posizionale

- In generale,
 - dato un numero $B \geq 2$, detto **BASE**, e
 - dato l'insieme composto da B *simboli diversi* $\beta = \{0, \dots, B-1\}$,
 - la stringa di n cifre:
 $\mathbf{b_{n-1}b_{n-2} \dots b_1b_0}$ con b_i appartenente a β
 - si interpreta *come*:
$$\mathbf{b_{n-1} \times B^{n-1} + b_{n-2} \times B^{n-2} + \dots + b_1 \times B^1 + b_0 \times B^0}$$

Esempi con basi diverse

- 217_8
- 36_{16}
- $2A_{16}$
- 1001_2

Basi più interessanti per i calcolatori

- Base 2 e base 16.

B=10	B=2	B=16
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	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
16	10000	10
17	10001	11
18	10010	12

Numeri binari naturali

- *Numeri binari naturali*: si utilizza un sistema di numerazione posizionale in base $B = 2$

- La sequenza di bit:

$$b_n b_{n-1} \dots b_1$$

dove si ha $b_j = 0$ o 1 , in base 2 rappresenta:

$$b_n \times 2^{n-1} + b_{n-1} \times 2^{n-2} + \dots + b_1 \times 2^0$$

- Esempi con $n = 8$:

- $00000000_2 = 0_{10}$

- $00001000_2 = 1 \times 2^3 = 8_{10}$

- $00101011_2 = 1 \times 2^5 + 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 = 43_{10}$

- $11111111_2 = \sum_{n=1,2,3,4,5,6,7,8} 1 \times 2^{n-1} = 255_{10}$

Conversioni tra basi

- da base 2 a base 10
- da base 10 a base 2
- da base 2 a base 16
- da base 16 a base 2
- tra basi generiche

Conversione da base 2 a base 10

- Considerando la relazione:

$$b_{n-1} \times B^{n-1} + b_{n-2} \times B^{n-2} + \dots + b_1 \times B^1 + b_0 \times B^0$$

che nel caso di $B=2$ diviene:

$$b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \dots + b_1 \times 2^1 + b_0 \times 2^0$$

è sufficiente sommare le potenze corrispondenti alle cifre '1'. Esempio:

- **7 6 5 4 3 2 1 0**

$$01001101 = 2^6 + 2^3 + 2^2 + 2^0 = 64 + 8 + 4 = 77_{10}$$

Conversione *rapida* da base 2 a base 10 (1/2)

- In binario si definisce una *notazione abbreviata*, sulla falsariga del sistema metrico-decimale:

$$k = 2^{10} = 1.024 \approx 10^3 \quad (\text{kilo})$$

$$M = 2^{20} = 1.048.576 \approx 10^6 \quad (\text{Mega})$$

$$G = 2^{30} = 1.073.741.824 \approx 10^9 \quad (\text{Giga})$$

$$T = 2^{40} = 1.099.511.627.776 \approx 10^{12} \quad (\text{Tera})$$

- Attenzione: K, M, G e T in base 2 abbiano valori molto prossimi ai corrispondenti simboli del sistema metrico-decimale (sistema MKS).
- L'errore risulta $< 10\%$ (vedi come la 2^a cifra sia = 0).

Conversione *rapida* da base 2 a base 10 (2/2)

- Diventa molto facile e quindi *rapido* calcolare il valore *decimale approssimato* delle *potenze di 2*, anche se hanno esponente grande.

- Infatti è sufficiente:

- *tenere a mente* l'elenco dei valori esatti delle prime dieci potenze di 2:

$$2^0 = 1, \quad 2^1 = 2, \quad 2^2 = 4, \quad 2^3 = 8, \quad 2^4 = 16, \\ 2^5 = 32, \quad 2^6 = 64, \quad 2^7 = 128, \quad 2^8 = 256, \quad 2^9 = 512;$$

- e *scomporre* in modo *additivo* l'esponente in contributi di valore 10, 20, 30 o 40, "leggendoli" come successioni di simboli K, M, G oppure T.

Esempio:

$$2^{17} = 2^{7+10} = 2^7 \times 2^{10} = 128 \text{ K}$$

che può essere letto "128 mila"

in realtà, $2^{17} = 131.072$, errore = $1-128.000/131.072 \approx 2,3\%$

Conversione da base 10 a base 2 (1/3)

■ In generale,

- per convertire un numero decimale **N**
- in un numero binario (**B = 2**)
- occorre trovare la stringa di n cifre binarie:

$$\mathbf{b_{n-1}b_{n-2} \dots b_1b_0}$$

- tale che

$$\mathbf{N = b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \dots + b_1 \times 2^1 + b_0 \times 2^0}$$

Conversione da base 10 a base 2 (2/3)

$$N/2 = b_{n-1} \times 2^{n-2} + b_{n-2} \times 2^{n-2} + \dots + b_1 \times 2^0, \quad R=b_0$$

$$(b_{n-1} \times 2^{n-2} + b_{n-2} \times 2^{n-2} + \dots + b_1 \times 2^0)/2 =$$
$$b_{n-1} \times 2^{n-3} + b_{n-2} \times 2^{n-4} + \dots + b_2 \times 2^0, \quad R=b_1$$

...

la ricerca dei coefficienti b-i richiede che si prosegua il procedimento fino a che il quoziente ottenuto non è più divisibile.

Alla fine: $Q=b_{n-1}$ e $R=b_{n-2}$

La rappresentazione binaria si ottiene scrivendo da sn. verso dx. **i resti in ordine inverso** a quello secondo cui sono stati prodotti.

Conversione da base 10 a base 2 (3/3)

In pratica:

- 1. decidere se il numero sia pari (resto 0) oppure dispari (resto 1), e annotare il resto**
- 2. dimezzare il numero (trascurando il resto)**
- 3. ripartire da (1) fino a ottenere 1 oppure 0**

si calcolano i resti delle divisioni per due

$$19 : 2 \rightarrow 1$$

$$9 : 2 \rightarrow 1$$

$$4 : 2 \rightarrow 0$$

$$2 : 2 \rightarrow 0$$

$$1 : 2 \rightarrow 1$$

$$19_{10} = 10011_2$$

Esempio di convers. da base 10 a base 2

$35_{10} \rightarrow$

$$35 / 2 = 17, \quad R_0 = 1 ;$$

$$17 / 2 = 8, \quad R_1 = 1 ;$$

$$8 / 2 = 4, \quad R_2 = 0 ;$$

$$4 / 2 = 2, \quad R_3 = 0 ;$$

$$2 / 2 = 1 = Q, \quad R_4 = 0 \quad (1 \text{ non è pi\`u divisibile per } 2)$$

$\rightarrow 100011_2$

Conversione da binario ad esadecimale

- Si scompone il numero binario in gruppi di 4 bit e si traduce ogni gruppo nella corrispondente cifra esadecimale.

Esempio:

1100 0011 0101 → C35₁₆

Base 8 (*ottale*)

- Si usano solo le cifre 0-7 (scartando 8 e 9)
 - per esempio:

$$534_{\text{oct}} = 5 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 = 348_{10}$$

Base 16 (esadecimale)

- Si usano le cifre 0-9 e le lettere A-F (ponendo: A = 10, B = 11, ..., F = 15)

- per esempio:

$$\begin{aligned} \text{B7F}_{\text{hex}} &= \text{B} \times 16^2 + 7 \times 16^1 + 15 \times 16^0 = \\ &= 11 \times 16^2 + 7 \times 16^1 + 15 \times 16^0 = 2943_{10} \end{aligned}$$

Conversione da tra esadecimale e binario

- Si traduce ogni cifra esadecimale nella corrispondente codifica binaria.

Esempio: considerato il numero A03B_{16} traducendo ogni sua cifra in binario:

- A = 1010
- 0 = 0000
- 3 = 0011
- B = 1011

Quindi:

$$\text{A03B}_{16} \rightarrow 1010 \ 0000 \ 0011 \ 1011$$

Esempi di conversione esadec./binario

- Conversione: 010011110101011011_2 in hex

$$\begin{aligned} & \mathbf{00}01_2 \quad 0011_2 \quad 1101_2 \quad 0101_2 \quad 1011_2 = \\ & = 1_{\text{dec}} \quad 3_{\text{dec}} \quad 13_{\text{dec}} \quad 5_{\text{dec}} \quad 11_{\text{dec}} = \\ & = 1_{\text{hex}} \quad 3_{\text{hex}} \quad \mathbf{D}_{\text{hex}} \quad 5_{\text{hex}} \quad \mathbf{B}_{\text{hex}} = \\ & = \mathbf{13D5B}_{\text{hex}} \end{aligned}$$

- Conversione: $A7B40C_{\text{hex}}$ in binario

$$\begin{aligned} & \mathbf{A}_{\text{hex}} \quad 7_{\text{hex}} \quad \mathbf{B}_{\text{hex}} \quad 4_{\text{hex}} \quad 0_{\text{hex}} \quad \mathbf{C}_{\text{hex}} = \\ & = 10_{\text{dec}} \quad 7_{\text{dec}} \quad 11_{\text{dec}} \quad 4_{\text{dec}} \quad 0_{\text{dec}} \quad 12_{\text{dec}} = \\ & = 1010_2 \quad 0111_2 \quad 1011_2 \quad 0100_2 \quad 0000_2 \quad 1100_2 = \\ & = 101001111011010000001100_2 \end{aligned}$$

Conversione tra base B^k e base B

- Esercizio

Aritmetica binaria dei numeri naturali

- Aumento
- Riduzione
- Somma
- Sottrazione
- Moltiplicazione
- Divisione

Aumento e riduzione di bit

- *Aumento* dei bit: **premettendo in modo progressivo un bit 0 a sinistra, il valore del numero non muta.** Esempio:

$$4_{10} = 100 = 0100 = 00100 = \dots$$

$$5_{10} = 101 = 0101 = 00101 = \dots$$

- *Riduzione* dei bit: **cancellando in modo progressivo un bit 0 a sinistra, il valore del numero non muta, ma bisogna arrestarsi quando si trova un bit 1!** Esempio:

$$7_{10} = 000111 = 00111 = 0111 = 111 \quad \text{STOP!}$$

$$2_{10} = 0010 = 010 = 10 \quad \text{STOP!}$$

Aritmetica binaria: somma numeri naturali

■ Tabella della somma:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 0$ e riporto di 1

■ 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 $n=8$:

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!

Aritmetica binaria: sottrazione numeri naturali

- Tabella della sottrazione:
- Esempio: 1010-111

- 0 - 0 = 0
- 0 - 1 = 1 e prestito di 1
- 1 - 0 = 1
- 1 - 1 = 0

$$\begin{array}{r}
 11 \\
 1010- \\
 111 \\
 \hline
 0011
 \end{array}$$

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

catena di propagazione dei prestiti

$$\begin{array}{r}
 \textcircled{11} \\
 (9) \ 1001 - \\
 (3) \ 0011 = \\
 \hline
 (6) \ 0110
 \end{array}$$



differenza (n bit)

catena di propagazione dei prestiti

$$\begin{array}{r}
 \textcircled{1111} \\
 (2) \ 0010 - \\
 (3) \ 0011 = \\
 \hline
 (?) \ 11111
 \end{array}$$



prestito in uscita

Aritmetica binaria: moltiplicazione numeri naturali

■ Tabella della moltiplicazione:

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

■ Esempio: 1010×111

$$\begin{array}{r}
 1010x \\
 111 \\
 ----- \\
 1010+ \\
 1010 + \\
 1010 \\
 ----- \\
 1000110
 \end{array}$$

Aritmetica binaria: divisione

■ Tabella della divisione:

- $0 : 1 = 0$
- $1 : 1 = 1$

■ Esempio: $1100:11$

```
1100:11
 000 | 100
    00 |
      0 |
```

■ Esempio: $10101:11$

```
10101:11
  100
   011
    0
```

Numeri relativi

- In un calcolatore i numeri sono rappresentati in gruppi di bit o parole (es. 8, 16, 32, ...)
- Sia $\mathbf{B} = \mathbf{b}_{n-1}\mathbf{b}_{n-2} \dots \mathbf{b}_1\mathbf{b}_0$
con b_i appartenente a $\beta = \{0,1\}$
- B rappresenta 2^n numeri interi positivi: $0 < B < 2^{n-1}$
- stabiliamo una convenzione:
 - se **MSB = 1** $\rightarrow \mathbf{B} < 0$
 - se **MSB = 0** $\rightarrow \mathbf{B} \geq 0$, ove MSB (*Most Significant Bit*) = b_{n-1}

Numeri relativi: rappresentazioni

- Esistono tre rappresentazioni principali:
 - **Modulo e segno**
 - **Rappresentazione in complemento a 1**
 - **Rappresentazione in complemento a 2**

Rappresentazione modulo e segno (1/3)

- 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 $2^{n-1}-1$

Nota.

Il numero 0 esiste sia come $+0$ che come -0 .

Rappresentazione modulo e segno (2/3)

- ***Numeri binari interi*** (positivi e negativi) ***in modulo e segno***:
 - 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 (3/3)

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.

Rappresentazione in 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:

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

$$-5_{10} \rightarrow 1010_2$$

Rappresentazione in complemento a 2 (1/2)

- 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 complementando a 1 e quindi sommando 1 al numero ottenuto.

- Esempio per $n=4$ bit:

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

$$-5_{10} \rightarrow 1010_2 + 0001_2 = 1011_2$$

Rappresentazione in complemento a 2 (2/2)

- Nella rappresentazione in complemento a 2 (C_2) il bit il più significativo ha *peso negativo*, mentre tutti gli altri bit hanno peso positivo.

- La sequenza di bit:

$$b_n b_{n-1} \dots b_1$$

rappresenta in C_2 il valore:

$$-b_n \times 2^{n-1} + b_{n-1} \times 2^{n-2} + \dots + b_1 \times 2^0$$

- Il bit più a sinistra viene chiamato *bit di segno*.

Esempio: numero a 3 bit in complemento a 2

- $000_{C_2} = -0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 0_{10}$
- $001_{C_2} = -0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1_{10}$
- $010_{C_2} = -0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2_{10}$
- $011_{C_2} = -0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 2 + 1 = 3_{10}$
- $100_{C_2} = -1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = -4_{10}$
- $101_{C_2} = -1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = -4 + 1 = -3_{10}$
- $110_{C_2} = -1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = -4 + 2 = -2_{10}$
- $111_{C_2} = -1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = -4 + 2 + 1 = -1_{10}$

Invertire un numero in Complemento a 2

- L'*inverso additivo* (od *opposto*) di un numero rappresentato in C_2 si ottiene:
 - invertendo (negando) ogni bit del numero
 - addizionando 1 nella posizione meno significativa
- Esempio per $n=5$:

$$\begin{aligned}01011_{C_2} &= 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 = 8 + 2 + 1 = 11_{10} \\10100 + 1 &= 10101_{C_2} = -1 \times 2^4 + 1 \times 2^2 + 1 \times 2^0 = \\ &= -16 + 4 + 1 = -11_{10}\end{aligned}$$

Conversione da Decimale a C_2

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

Aumento e Riduzione dei Bit in C_2

- *Estensione del segno: replicando* in modo progressivo il bit di segno a sinistra, il valore del numero non muta. Per esempio:
 $4 = 0100 = 00100 = 000100 = \dots$ (indefinitamente)
 $-5 = 1011 = 11011 = 111011 = \dots$ (indefinitamente)
- *Contrazione del segno: 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)

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 ...
- 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) !

Riepilogo rappresentazione interi su 4 bit

Dec.	Segno e modulo	Compl. a 1	Compl. a 2	Dec.	Segno e modulo	Compl. a 1	Compl. a 2
7	0111	0111	0111	-0	1000	1111	-
6	0110	0110	0110	-1	1001	1110	1111
5	0101	0101	0101	-2	1010	1101	1110
4	0100	0100	0100	-3	1011	1100	1101
3	0011	0011	0011	-4	1100	1011	1100
2	0010	0010	0010	-5	1101	1010	1011
1	0001	0001	0001	-6	1110	1001	1010
0	0000	0000	0000	-7	1111	1000	1001
				-8	-	-	1000

Confronto tra Rappresentazioni

<i>B</i>				valore rappresentato			
<i>b</i> ₃	<i>b</i> ₂	<i>b</i> ₁	<i>b</i> 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

Esempio di addizione algebrica 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}

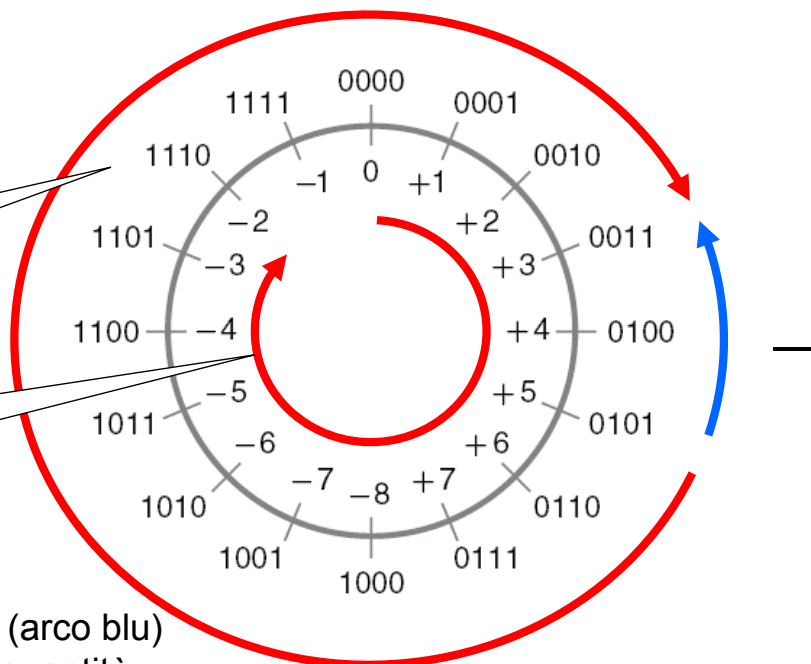
Come funziona l'addizione algebrica in C_2

struttura della
rappresentazione
(per $n = 4$)

14
(in naturale)

+

identico ad
arco rosso
esterno



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

Trabocco e overflow nell'addizione in C_2

Pesi	7	6	5	4	3	2	1	0		
Riporto	1		1	1	1					
Addendo 1	0	1	0	0	1	1	0	1	+	77_{dec}
Addendo 2	0	1	0	1	1	1	0	0	=	92_{dec}
Somma	1	0	1	0	1	0	0	1		$-87_{dec}!$

trabocco (overflow)
risultato negativo!

risultato errato!

addizione algebrica con trabocco

Come rilevare in trabocco in C_2

- 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 algebrica in C_2

- Per sottrarre in complemento a due, basta prima invertire (in C_2) il sottraendo, e poi addizionare.
- Salvo il verificarsi di trabocco, la sottrazione in C_2 non ha restrizioni.
- In effetti, in C_2 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.

Esempio sottrazione usando il C_2 (caso 1)

- $n=8$, $30 - 22$

$$30_{10} \rightarrow 00011110_2$$

$$22_{10} \rightarrow 00010110_2, \text{ trasformo } 22 \text{ in compl. a } 2:$$

$$-22_{10} \rightarrow 11101001_2 + 1 = 11101010_2$$

Ora sommo a 30 il -22 così ottenuto:

$$30_{10} \rightarrow 00011110_2 +$$

$$-22_{10} \rightarrow 11101010_2 =$$

$$100001000$$

↑ c'è riporto! → il risultato è positivo.

Esempio sottrazione usando il C_2 (caso 2)

- $n=8, 19 - 22$

$$19_{10} \rightarrow 00010011_2$$

$$22_{10} \rightarrow 00010110_2, \text{ trasformo } 22 \text{ in compl. a } 2:$$

$$-22_{10} \rightarrow 11101001_2 + 1 = 11101010_2$$

Ora sommo a 19 il -22 così ottenuto:

$$19_{10} \rightarrow 00010011_2 +$$

$$-22_{10} \rightarrow 11101010_2 =$$

$$\begin{array}{r} \text{-----} \\ _11111101 = -3_{10} \end{array}$$

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

Esempi vari di addizione algebrica in C_2

$$\begin{array}{r} 0010 + (+2) \\ 0011 = (+3) \\ \hline \end{array}$$

$$0101 \quad (+5)$$

$$\begin{array}{r} 1011 + (-5) \\ 1110 = (-2) \\ \hline \end{array}$$

$$1001 \quad (-7)$$

$$\begin{array}{r} 1101 - (-3) \\ 1001 = (-7) \\ \hline \end{array}$$

$$\begin{array}{r} 0010 - (+2) \\ 0100 = (+4) \\ \hline \end{array}$$



$$\begin{array}{r} 0100 + (+4) \\ 1010 = (-6) \\ \hline \end{array}$$

$$1110 \quad (-2)$$

$$\begin{array}{r} 0111 + (+7) \\ 1101 = (-3) \\ \hline \end{array}$$

$$0100 \quad (+4)$$

$$\begin{array}{r} 1101 + \\ 0111 = \\ \hline \end{array}$$

$$0100 \quad (+4)$$

$$\begin{array}{r} 0010 + \\ 1100 = \\ \hline \end{array}$$

$$1110 \quad (-2)$$

Osservazioni sulla rappresentazione in C_2

- 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

Osservazioni sulle operazioni in C_2

- 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.
- L'aritmetica del calcolatore è una scienza importante (e difficile), cui il calcolatore ha dato un impulso decisivo negli ultimi 60 o 70 anni.

Numeri razionali

- Rappresentazione in virgola fissa
- Rappresentazione in virgola mobile (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: da binario a decimale

- La stringa: $b_{-1}b_{-2} \dots b_{-m}$ si interpreta come:

$$b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + \dots + b_{-m} \times 2^{-m}$$

- Esempi:

- $0,101_2$ corrisponde al numero decimale:

$$1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} =$$

$$1/2 + 1/8 =$$

$$0,5 + 0,125 = 0,625_{10}$$

- $0,1011_2$ corrisponde al numero decimale:

$$1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} =$$

$$1/2 + 1/8 + 1/16 =$$

$$0,5 + 0,125 + 0,0625 =$$

$$0,6875_{10}$$

Numeri frazionari: da decimale a binario

- Dato il numero F in base 10, si tratta di trovare la stringa $b_{-1}b_{-2} \dots b_{-m}$ tale per cui:

$$F = b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + \dots + b_{-m} \times 2^{-m}$$

- Osservando che b_{-1} è la parte intera del prodotto:
 $2 \times F = b_{-1} + b_{-2} \times 2^{-1} + \dots + b_{-m} \times 2^{-(m-1)}$

Si deduce che la ricerca dei coefficienti b_i richiede un processo di successive moltiplicazioni della parte frazionaria con la parte intera.

Il processo termina solo quando la parte frazionaria risulta 0 oppure il numero è periodico.

Esempio di numeri frazionari in binario

- $0,78125_{10} \rightarrow$
 $0,78125 \times 2 = 1,5625 \rightarrow 1$
 $0,5625 \times 2 = 1,125 \rightarrow 1$
 $0,125 \times 2 = 0,250 \rightarrow 0$
 $0,25 \times 2 = 0,5 \rightarrow 0$
 $0,5 \times 2 = 1,0 \rightarrow 1$
dunque: $0,78125_{10} = 0,11001_2$
- $19,6875_{\text{dec}} = 10011,1011$ in quanto
 $19_{\text{dec}} = 10011_2$ e $0,6875_{\text{dec}} = 0,1011_2$
5 bit parte intera, 4 bit parte frazionaria

Osservazioni sulla 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 (1/2)

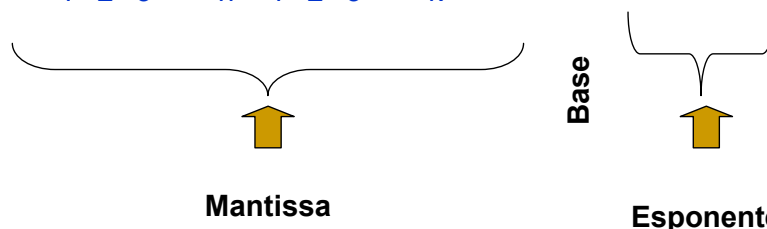
- Per rappresentarli si fa uso di rappresentazioni normalizzate aventi lo scopo di non doversi curare della posizione della virgola e del numero delle cifre utilizzate.
- Queste rappresentazioni vengono espresse come il prodotto di due fattori:
 - le cifre significative del numero
 - Una potenza del 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*:

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

Numeri in virgola mobile (2/2)

- In generale una rappresentazione approssimata di un numero è data da:

$$\pm x_1 x_2 x_3 \dots x_h, y_1 y_2 y_3 \dots y_k \times B^{\pm a_1 a_2 \dots a_n}$$



- In binario, si utilizzano gli elementi seguenti:
 - un bit s per esprimere il segno
 - $m \geq 1$ bit per la *mantissa* (numero naturale)
 - $e' \geq 1$ bit per l'*esponente* (numero intero)in totale dunque occorrono $1 + m + e'$ bit.

Rappresentazione esponenziale normalizzata

- *Convenzione:*
la prima cifra significativa si trova immediatamente a destra della virgola.
- 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: $127 \times 10^6 = 0,127 \times 10^9$
 $15 \times 10^{-7} = 0,15 \times 10^{-5}$

Compattamento dalla rappresentazione

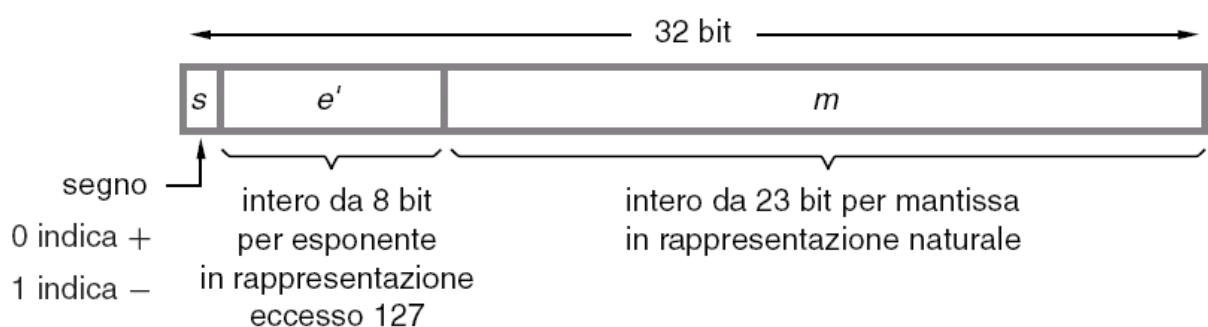
- Eliminare:
 - 0 iniziale
 - la virgola decimale
 - il segno di prodotto
 - il valore della base della potenza
- Lunghezza della mantissa costante
- Limitare gli esponenti ad un intervallo
- Polarizzare l'esponente
- Disporre segno, esponente e mantissa in un ordine stabilito.

Standard IEEE-754 (formato base, precisione singola)

- La mantissa è un numero la cui parte intera è sempre 1, ma il corrispondente bit è nascosto (la mantissa si compone solo della parte frazionaria).
- 1 bit di segno, 8 di esponente e 23 di mantissa.

Ampiezza parola	32
Ampiezza esponente	8
P (mantissa)	23
E_{\max}	127
E_{\min}	-126
Costante polarizzazione	127

Formato IEEE-754 singola precisione



- Il valore (frazionario) del numero rappresentato è:

$$\text{val}(se'm) = \pm f \times 2^e = \begin{cases} +1, m \times 2^{e'-127} & \text{se } s = 0 \\ -1, m \times 2^{e'-127} & \text{se } s = 1 \end{cases}$$

dove $f = 1, m$ ed $e = e' - 127$

Esempi (1/3)



$$e' = 32 + 8 = 40$$

$$\text{valore rappresentato} = \pm 1,001010 \dots 0 \times 2^{40-127} = +1,001010 \dots 0 \times 2^{-87}$$

Esempi (2/3)

- $5_{10} = 101_2 = 1,01_2 \times (10^{10})_2$
 - il segno è positivo,
 - l'esponente vale $(2_{10} + \text{costante di polarizzazione}) = 129_{10}$ cioè 10000001_2
 - la mantissa vale 01

dunque la rappresentazione e':

s *esponente* *mantissa*

0 10000001 010000000000000000000000

Esempi (3/3)

- $1284,91_{10} = 1010000\ 0100,1110100011110\dots_2 =$
- $1, 01000001001110100011110_2 \times (10^{1010})_2$
 - il segno è positivo,
 - l'esponente vale $(10_{10} + \text{costante di polarizzazione}) = 137_{10}$ cioè 10001001_2
 - la mantissa vale 01000001001110100011110

dunque la rappresentazione è:

s *esponente* *mantissa*

0 10001001 01000001001110100011110

Osservazioni sulla rappresentazione in virgola mobile

- 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.

Operazioni in virgola mobile

- Il formato di virgola mobile dispone di algoritmi abbastanza semplici ed efficienti per calcolare addizione, sottrazione e moltiplicazione, e altre operazioni ancora.
- Le operazioni in virgola mobile vengono ridotte a (poche) operazioni su numeri interi.
- Nel fare questo, ci sono numerosi aspetti importanti da definire: come arrotondare, se trattare o meno gli errori, e simili.

Operazioni in virgola mobile

- Addizione e sottrazione
 - Si rendono eguali gli esponenti mediante la traslazione a dx del numero con esponente minore (l'esponente del risultato è uguale all'esponente del più grande)
 - Addizione delle mantisse (e segno)
 - Eventuale normalizzazione
- Moltiplicazione
 - Somma esponenti e sottrazione costante di polarizzazione
 - Moltiplicazione delle mantisse (e segno)
 - Eventuale normalizzazione
- Divisione
 - Sottrazione esponenti e somma costante di polarizzazione
 - Moltiplicazione delle mantisse (e segno)
 - Eventuale normalizzazione

Precisione: bit di guardia

- I registri dei calcolatori moderni hanno dimensioni maggiori di quelle della mantissa
- **Bit di guardia:** sono i bit rimanenti del registro non presenti nella mantissa ed utilizzati nei passaggi intermedi dell'elaborazione (maggior precisione di calcolo).
- Il numero copiato in memoria viene arrotondato ed i bit di guardia vengono eliminati.

Precisione: Arrotondamento

- Standard IEEE 754-1985
 - Arrotondamento standard verso il valore più vicino
 - Esempio:
 - $0, b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8$
 - Arrotond. $\rightarrow 0, b_1 b_2 b_3 b_4$
 - Se è $\geq 1000_2 \rightarrow$ si aggiunge 1 all'ultima cifra (b_4)
 - Se è $< 1000_2 \rightarrow$ si tronca .
 - Troncamento
 - ! polarizzazione verso lo zero
 - Arrotondamento per eccesso
 - Arrotondamento per difetto

Eccezioni

- Vengono segnalate dalla CPU al programmatore al fine di poterle gestire.
 - Underflow
 - Overflow
 - Divisione per zero
 - Eccezione per inesattezza
 - Eccezione per invalidità

Informazioni di carattere alfanumerico

■ Codifica ASCII

E' una forma di codifica alfanumerica, attualmente e' su 8 bit e consente di rappresentare tutti i caratteri alfabetici, maiuscoli e minuscoli, i numeri, i segni di punteggiatura, alcuni simboli matematici oltre ad alcuni *caratteri di controllo*.

Esempio: 'A' = $41_{16} = 65_{10}$, '9' = $39_{16} = 57_{10}$

■ Codifica BCD (*Binary Coded Decimal*)

Viene usata per rappresentare le cifre binarie decimali su 4 bit.

Esempio: $147_{10} = 0001\ 0100\ 0111_{BCD}$

Tabella ASCII standard

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL	32	20	!	64	40	@	96	60	`
1	01	SOH	33	21	"	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	\$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(72	48	H	104	68	h
9	09	TAB	41	29)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	
29	1D	GS	61	3D	=	93	5D]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL