

Tipologie di codici

Nel seguito vedremo tipologie di rappresentazioni diverse:

- Senza assumere limitazioni sul numero di bit a disposizione:
per numeri [notazione binaria, ovvero posizionale con base 2]
- Disponendo di un numero di bit limitato:
 - numeri naturali
 - interi relativi [valore assoluto e segno, complemento a due]
 - “reali” [virgola fissa e virgola mobile]
 - valori logici, caratteri alfabetici, testi
 - suoni, immagini e sequenze video
 - codici per la rilevazione e correzione di errori
- Codici di compressione (senza | con perdita)

Codifica di numeri naturali

I numeri naturali si rappresentano normalmente, ma con n cifre binarie possiamo rappresentare solo i numeri da 0 a N_{max}

Esempio: con 8 cifre ($n=8$)

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

1

...

0	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

127

...

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

$N_{max} = 255$

Nota

Quanto vale il numero binario

$\underbrace{111 \dots 1}_{n \text{ cifre}}$

?

Nota

Quanto vale il numero binario

$$\underbrace{111 \dots 1}_{n \text{ cifre}}$$

?

Notare che

$$\begin{array}{r} 111 \dots 1 + \\ \hline 1 = \\ 1 \ 000 \dots 0 \end{array}$$

Nota

Quanto vale il numero binario

$\underbrace{111 \dots 1}_{n \text{ cifre}}$

?

Notare che

$$\begin{array}{r} 111 \dots 1 + \\ \hline 1 = \\ 1 \ 000 \dots 0 \end{array} \quad \leftarrow 2^n$$

Quindi

$\underbrace{111 \dots 1}_{n \text{ cifre}} \quad \text{vale } 2^n - 1$

Quindi

Con n cifre binarie si possono rappresentare i numeri da 0 a 2^n-1

Esempio precedente

$$n = 8$$

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

$$N_{\max} = 2^n - 1 = 256 - 1 = 255$$

Viceversa

Voglio rappresentare i numeri naturali da 0 a N_{\max} .

Di quante cifre binarie ho bisogno?

Esempio

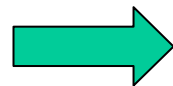
Voglio rappresentare numeri da 0 a 350

...

con $n = 7$ $N_{\max} = 127$

con $n = 8$ $N_{\max} = 255$

con $n = 9$ $N_{\max} = 511$

 $n = 9$

In generale

Per poter rappresentare numeri naturali fino a $N \geq 0$, serve un numero di cifre n tali che:

$$N_{\max} \geq N \quad \text{ovvero} \quad (2^n - 1) \geq N$$

Quindi deve essere

$$n \geq \log_2(N + 1)$$

Esempio precedente

$$N = 350$$

$$n \geq \log_2(351) = 8, \dots$$

$$\text{quindi } n \geq 9$$

OPERAZIONI ARITMETICHE

Nel caso di addizione, ho traboccamento (overflow) quando ho un riporto dal bit più significativo che non può essere rappresentato con le cifre a disposizione.

$$\begin{array}{r} 1\ 1\ 1\ 0\ +\ 14 \\ 0\ 0\ 1\ 0\ =\ 2 \\ \hline (1)\ 0\ 0\ 0\ 0\ \quad 0? \end{array}$$

Nel caso di sottrazione, un prestito dal bit più significativo indica un risultato negativo (non rappresentabile).

$$\begin{array}{r} 1\ 1\ 0\ 0\ -\ 12 \\ 1\ 1\ 0\ 1\ =\ 13 \\ \hline (1)\ 1\ 1\ 1\ 1\ \quad 15? \end{array}$$

Tipologie di codici

Nel seguito vedremo tipologie di rappresentazioni diverse:

- Senza assumere limitazioni sul numero di bit a disposizione:
per numeri [notazione binaria, ovvero posizionale con base 2]
- Disponendo di un numero di bit limitato:
 - numeri naturali
 - interi relativi [valore assoluto e segno, complemento a due]
 - “reali” [virgola fissa e virgola mobile]
 - valori logici, caratteri alfabetici, testi
 - suoni, immagini e sequenze video
 - codici per la rilevazione e correzione di errori
- Codici di compressione (senza | con perdita)

Codifica in valore assoluto e segno

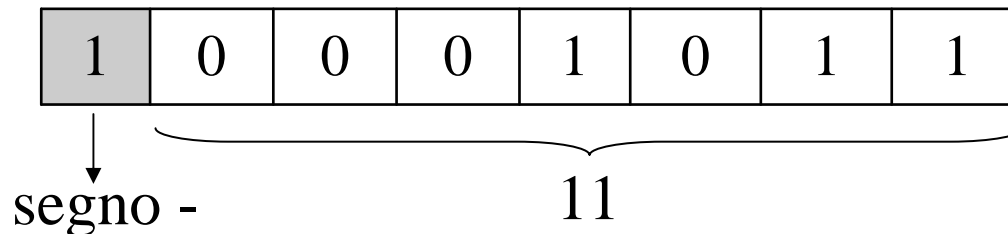
Avendo n bit a disposizione e un numero da rappresentare:

- il bit più significativo rappresenta il segno del numero
(0 = segno + 1 = segno -)
- Gli altri $n-1$ bit rappresentano il valore assoluto

Esempio

$n = 8$ (ho 8 bit a disposizione)

Rappresentazione di -11



Esempio: codifica con valore assoluto e segno con 4 bit

Numeri positivi		Numeri negativi	
0000	+0	1000	-0
0001	+1	1001	-1
0010	+2	1010	-2
0011	+3	1011	-3
0100	+4	1100	-4
0101	+5	1101	-5
0110	+6	1110	-6
0111	+7	1111	-7

Attenzione: 2 rappresentazioni dello 0!!!

Note

1) Esistono due codifiche per il valore 0

0	0	0	0	0	0	0	0	0^+
---	---	---	---	---	---	---	---	-------

1	0	0	0	0	0	0	0	0^-
---	---	---	---	---	---	---	---	-------

2) I valori rappresentabili vanno:

da $-(2^{n-1} - 1)$ a $+(2^{n-1} - 1)$

segno	*	*	*	*	*	*	*
-------	---	---	---	---	---	---	---

n-1 bit: da 0 a $2^{n-1} - 1$

Questa tecnica di rappresentazione non viene usata dal calcolatore

Difficoltà nel fare le operazioni aritmetiche, p.es. la sottrazione

Es. 1 (sottrazione di numeri entrambi positivi)

a:

0	0	0	1	1	0	1	1
---	---	---	---	---	---	---	---

 - 27

b:

0	0	1	0	1	0	1	1
---	---	---	---	---	---	---	---

 43

Dato che $|b| > |a|$, il segno del risultato è neg, il valore assoluto è

$$\begin{array}{r} 0101011 - |b| \\ 0011011 = |a| \\ \hline 0010000 \end{array}$$

➡ a-b:

1	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---

 -16

Es. 2 (sottrazione di numeri entrambi positivi)

a:

0	0	1	1	1	0	1	1
---	---	---	---	---	---	---	---

 - 59

b:

0	0	1	0	1	0	1	1
---	---	---	---	---	---	---	---

 43

Dato che $|a| > |b|$, il segno del risultato è pos, il valore assoluto è

$$\begin{array}{r} 0111011 - |a| \\ 0101011 = |b| \\ \hline 0010000 \end{array}$$

→ a-b:

0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---

 +16

In generale per la sottrazione

Segno a	Segno b		segno(a-b)	a-b
+	+	$ a > b $	+	$a - b$
+	+	$ b > a $	-	$b - a$
-	+		-	$ a + b$
+	-		+	$a + b $
-	-	$ a > b $	-	$ a - b $
-	-	$ b > a $	+	$ b - a $

Per il calcolatore le operazioni di somma e sottrazione sono complesse

➡ Si vuole una rappresentazione per la quale esista
un unico semplice metodo per l'addizione e la sottrazione...
vedremo la tecnica di codifica principale (complemento a due)

Codifica in complemento a due

Avendo a disposizione n bit:

**i numeri positivi sono rappresentati normalmente (rappresentazione binaria dei numeri positivi);
il bit più significativo è pari a 0**

**i numeri negativi si ottengono come complemento a 2 del numero positivo N corrispondente, ovvero come $(2^n - N)$;
il bit più significativo è pari a 1**

Codifica in complemento a due

Avendo a disposizione n bit:

**i numeri positivi sono rappresentati normalmente (rappresentazione binaria dei numeri positivi);
il bit più significativo è pari a 0**

**i numeri negativi si ottengono come complemento a 2 del numero positivo N corrispondente, ovvero come $(2^n - N)$;
il bit più significativo è pari a 1**

ci sono due regole semplici
per determinare il complemento



Complemento a 2

Dato un numero N rappresentato in base 2 da n cifre il suo complemento a 2 è dato da:

$$C_2 = 2^n - N$$

Esempio:

$$N_2 = 01010100 \quad (84)$$

$$\begin{array}{r} C_2 = \quad 100000000 - \\ \quad \quad 01010100 \\ \hline \quad \quad 10101100 \end{array}$$

Regole pratiche equivalenti:

- partendo dal bit meno significativo e procedendo verso sinistra, si lasciano immutati tutti i bit fino al primo 1 compreso, poi si invertono gli altri;
oppure
- si complementano tutti i bit (complemento a uno) e si aggiunge 1

ESEMPIO: vogliamo rappresentare in complemento a due con 4 bit
il numero decimale -6

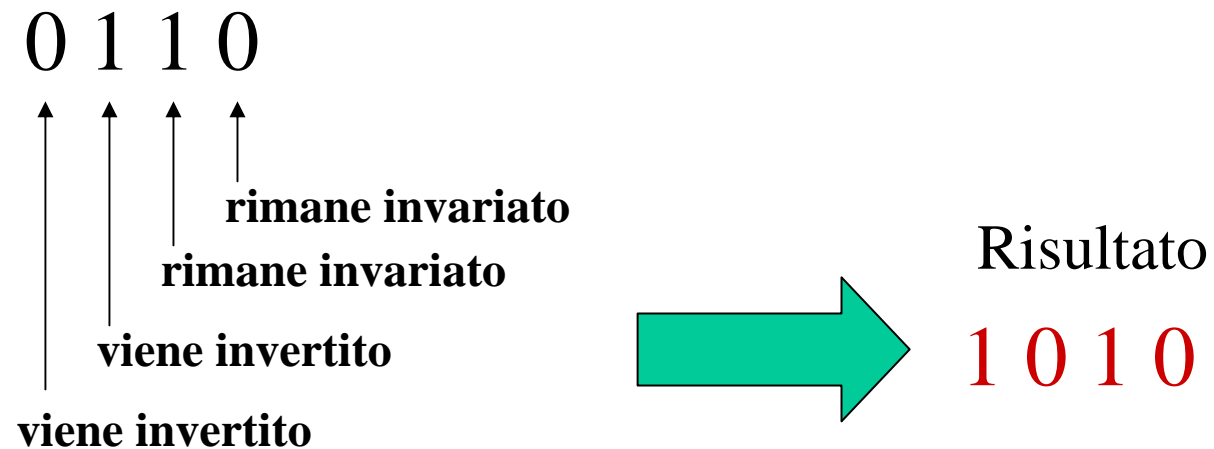
A) Utilizzando il complemento a uno

- Dato $+6_{\text{dieci}}$ codificato su 4 bit $\Rightarrow 0110$
- Facendo il complemento a 1 si ottiene 1001
- Sommando 1 al risultato si ottiene...

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

B) Facendo una conversione diretta:

- Dato $+6_{\text{dieci}}$ codificato su 4 bit $\Rightarrow 0110$



ESERCIZIO

Rappresentare in complemento a due con 8 bit il numero decimale -67 .

quindi

$$67_{10} = 01000011_2$$
$$-67 = 10111101$$

ERRORE TIPICO: dimenticare che la rappresentazione in C.a 2 è relativa ad un numero di bit fissati!!!

Es. Rappresentare in C. a 2 con 8 bit il numero decimale -3 .

$$3_{10} = 11_2$$

quindi $-3 = 01$

ma questo risulta un numero positivo!!!

ERRORE TIPICO: dimenticare che la rappresentazione in C.a 2 è relativa ad un numero di bit fissati!!!

Es. Rappresentare in C. a 2 con 8 bit il numero decimale -3 .

$$3_{10} = 11_2$$

quindi $-3 = 01$

ma questo risulta un numero positivo!!!

Svolgimento corretto:

$$3 = 00000011 \quad (8 \text{ bit})$$

$$-3 = 11111101$$

ALTRO ERRORE TIPICO: complementare sempre e comunque, anche i numeri positivi!

ES: Rappresentazione in complemento a due con 6 bit del numero decimale 15.

15 = 001111 anche codificato in compl. a 2!!!

Un esempio: codifica in complemento a due con 4 bit

Bit di segno	Numeri positivi	Bit di segno	Numeri negativi
→	0 000	→	1 000
	0		-8
	0 001		1 001
	1		-7
	0 010		1 010
	2		-6
	0 011		1 011
	3		-5
	0 100		1 100
	4		-4
	0 101		1 101
	5		-3
	0 110		1 110
	6		-2
	0 111		1 111
	7		-1

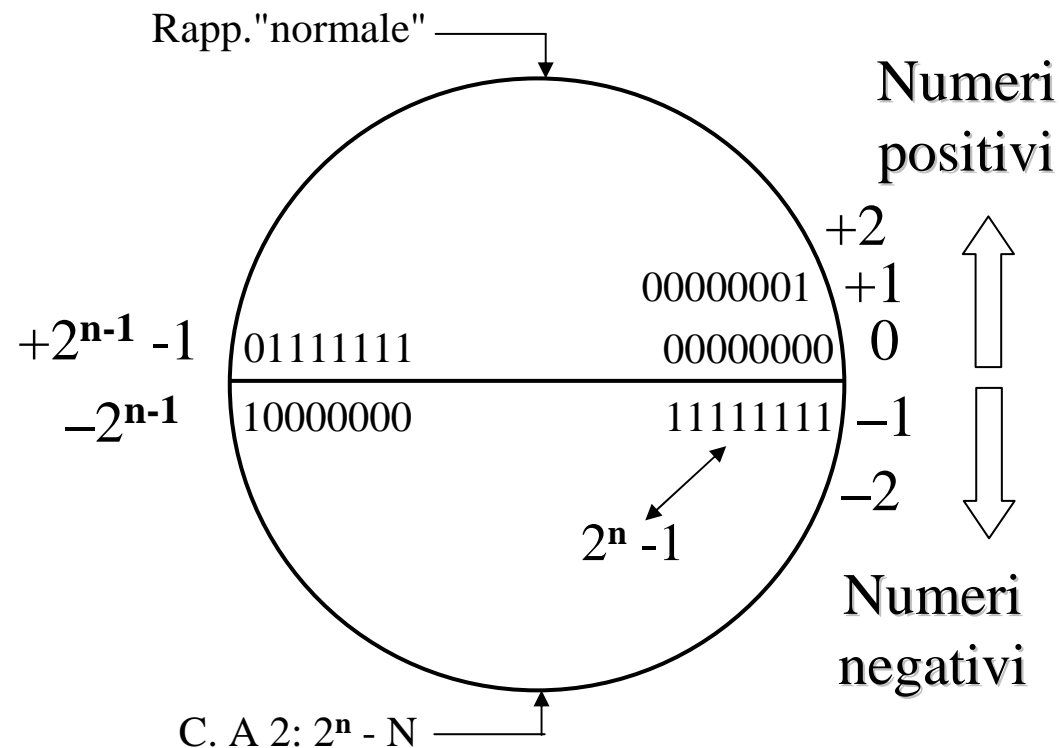
Unica rappresentazione del numero zero

NB: i numeri rappresentabili vanno da -2^{n-1} a $+(2^{n-1} - 1)$

Più in generale...

Idea: con n cifre si possono rappresentare 2^n numeri.

Metà per i positivi e metà per i negativi, come in figura



NB: i numeri rappresentabili vanno da -2^{n-1} a $+(2^{n-1} - 1)$

Esempio: intervalli di rappresentazione con $n=16$ bit

- *Rappresentazione in valore assoluto e segno*: numeri compresi fra $-(2^{15}-1)$ e $2^{15}-1$, ovvero fra **-32767 e +32767**... lo 0 ha due rappresentazioni
- *Rappresentazione in complemento a 2*: numeri compresi fra -2^{15} e $2^{15}-1$, ovvero fra **-32768 e +32767**... lo 0 ha una sola rappresentazione.

In pratica, però, tipicamente si utilizzano i valori fra **-32767 e +32767** per simmetria (in tal modo dato un qualsiasi numero anche il suo opposto è rappresentabile)

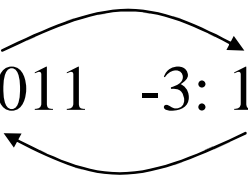
Conversione da codifica in complemento a due a decimale

- Si può usare la simmetria dell'operazione di complemento a due

Dato $p > 0$ rappresentato in C. A 2, per ottenere $-p$ ne faccio il C. A 2

Dato $p < 0$ rappresentato in C. A 2, per ottenere $-p > 0$ ne faccio il C. A 2

Es. $+3: 0011$ $-3: 1101$



- Esempio:

$1\ 0\ 0\ 0\ 0\ 0\ 1_{Ca2}$ è un numero negativo

quindi è pari a $-0111111_2 = -63_{10}$

Oppure si può usare la seguente regola...

- *Regola generale:*

il valore di un numero $c_{k-1}c_{k-2}\dots c_1c_0$ rappresentato in complemento a due è dato dalla seguente espressione

$$-c_{k-1}x2^{k-1} + c_{k-2}x2^{k-2} + \dots + c_1x2^1 + c_0x2^0$$

- Esempio:

$$1\ 0\ 0\ 0\ 0\ 0\ 1_{Ca2} = (-1)x2^6 + 1x2^0 = -64 + 1 = -63_{dieci}$$

Altri esempi

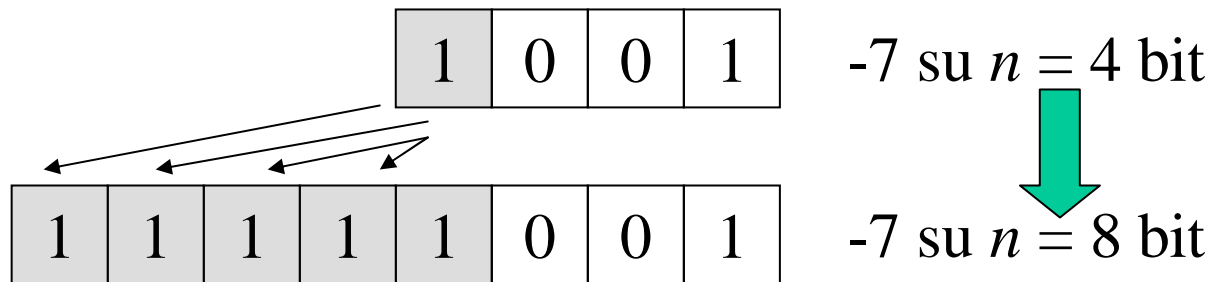
- $10001 = -1 \times 2^4 + 1 \times 2^0 = -16 + 1 = -15_{\text{dieci}}$
- $1010 = -1 \times 2^3 + 1 \times 2^1 = -8 + 2 = -6_{\text{dieci}}$
- $10101010 = -1 \times 2^7 + 1 \times 2^5 + 1 \times 2^3 + 1 \times 2^1 = -128 + 32 + 8 + 2 = -86$

Esercizio proposto:

ricavare il valore decimale col metodo della simmetria

Estensione del segno

Estendiamo il segno per rappresentare un numero su $n=k+d$ bit anziché su $n=k$ bit



Perché il complemento a 2?

- I **calcolatori** usano la rappresentazione in complemento a 2
 - si semplificano i circuiti che svolgono le operazioni aritmetiche
 - in particolare la **somma** si effettua semplicemente come nel caso di numeri naturali, inoltre
 - **somma** e **sottrazione** possono essere realizzate con un unico circuito: infatti: $x - y = x + (-y)$

Somma di numeri in complemento a 2

L'addizione di due numeri rappresentati in complemento a 2 dà un risultato corretto, *trascurando il riporto*, a patto che il *risultato sia compreso entro l'intervallo dei numeri rappresentabili*

Somma di numeri in complemento a 2

L'addizione di due numeri rappresentati in complemento a 2 dà un risultato corretto, *trascurando il riporto*, a patto che il *risultato sia compreso entro l'intervallo dei numeri rappresentabili*

Esempio

Usando $n = 6$ bit, l'intervallo dei numeri rappresentabili va da -2^5 a $+2^5-1$, ovvero da -32 a $+31$

Vogliamo calcolare $26 - 13$: $26 - 13 = 26 + (-13) = +13$

$$\begin{array}{r} 011010 + \\ 110011 = \\ \hline \end{array} \quad \begin{array}{r} 26 \\ -13 \end{array} \quad (13 = 001101)$$

$\boxed{1}001101$

↑
Il riporto viene trascurato

$+13$

↑
È nell'intervallo dei numeri rappresentabili

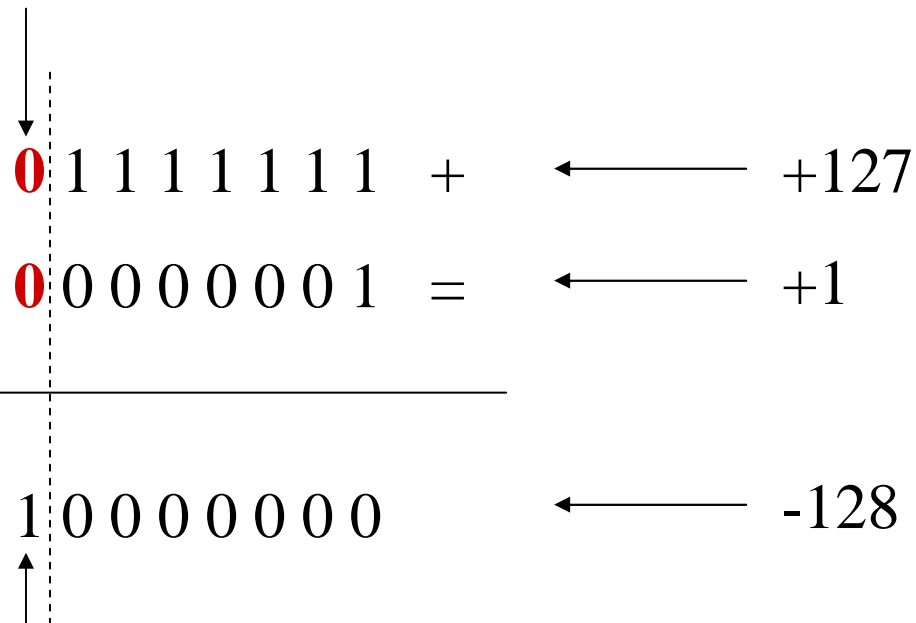
Overflow

- La somma di *due numeri interi positivi* o di *due numeri interi negativi* può dar luogo ad un intero non rappresentabile con i bit a disposizione
- Questo dà luogo a ciò che si chiama “overflow” (traboccamento)
- In caso di overflow, il risultato di un’operazione non è valido
- Esempio: supponiamo di avere a disposizione 8 bit per rappresentare gli interi.
Sommiamo a **01111111 (+127)** il numero **00000001 (+1)**
otteniamo un numero negativo **(-128)** invece di **+128**

NB: ma il calcolatore non “ragiona” in decimale...
come si può accorgere della presenza di overflow?

Esempio di overflow

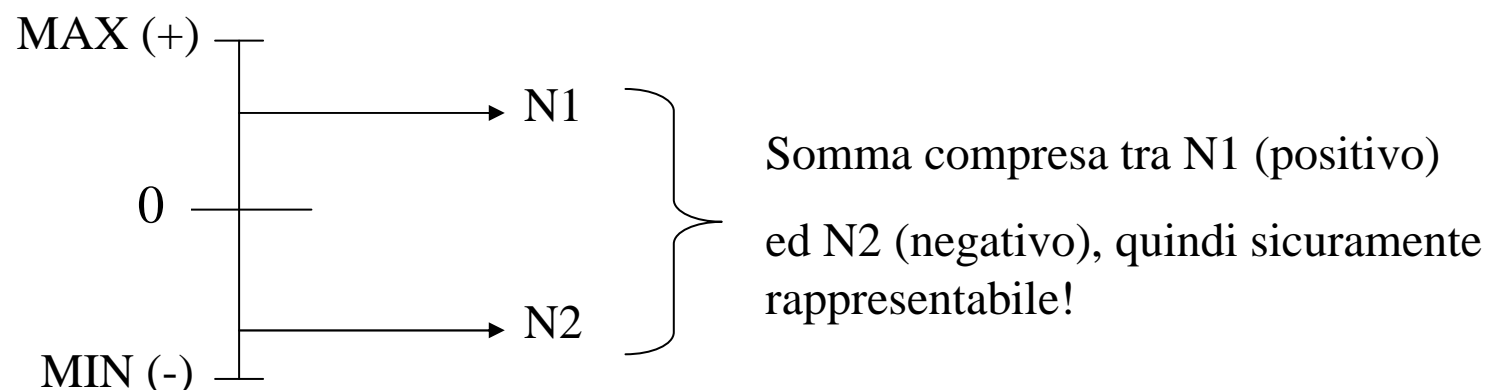
Bit di segno



**Il risultato ha segno negativo nonostante
gli addendi siano entrambi positivi**

Regola per la determinazione dell'overflow

- Se gli addendi hanno *segno discorde* non si ha mai overflow:



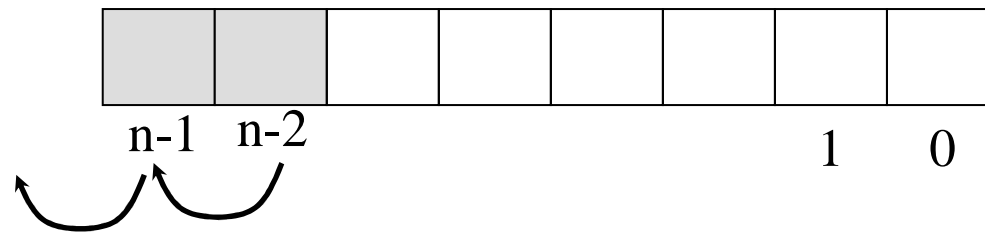
- Se gli addendi hanno *segno concorde* si ha overflow se **il segno del risultato è diverso dal segno dei due addendi:**

+ e +: deve risultare +, altrimenti overflow!

– e –: deve risultare –, altrimenti overflow!

Regola equivalente per l'overflow

- Con una rappresentazione su n bit, si ha *overflow* se i riporti generati nelle due posizioni più significative ($n-1$ e $n-2$ in figura) sono *diversi*




(ovvero: c'è riporto generato in una posizione ma non nell'altra)

Esempio di overflow

Usando $n = 6$ bit, l'intervallo dei numeri rappresentabili va da -2^5 a $+2^5-1$, ovvero da -32 a $+31$

Vogliamo calcolare $-25 - 13$

$-25 - 13 = -25 + (-13) = -38 \rightarrow$ non è compreso nell'intervallo

$$\begin{array}{rcl} 1\ 0\ 0\ 1\ 1\ 1 & + & -25 \quad (25 = 0\ 1\ 1\ 0\ 0\ 1) \\ \hline 1\ 1\ 0\ 0\ 1\ 1 & = & -13 \quad (13 = 0\ 0\ 1\ 1\ 0\ 1) \\ \hline \textcolor{red}{1}0\ 1\ 1\ 0\ 1\ 0 & & +26 \end{array}$$


**Generato un riporto solo nella
posizione più significativa**

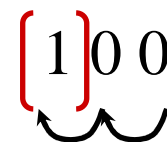
Esempio di non overflow

$25 - 13 = 25 + (-13) = 12 \rightarrow$ è compreso nell'intervallo

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

(13 = 0 0 1 1 0 1)

$\left[1 \right] 0\ 0\ 1\ 1\ 0\ 0\ +12$



**Generato un riporto
in entrambe le posizioni
più significative**

Esercizio (Appello del 23 set 2003)

Rappresentare i numeri -51 e -98 (in base 10) in notazione binaria in complemento a due con 8 bit. Eseguire la somma algebrica dei numeri così ottenuti e commentare il risultato. [3]

Soluzione

Conversione nella notazione binaria dei numeri (valori assoluti)

$$51 = 00110011_2$$

$(32 + 16 + 2 + 1)$

$$98 = 01100010_2$$


$(64 + 32 + 2)$

Rappresentazione in complemento a due:

$$-51 = 11001101$$

$$-98 = 10011110$$

Somma algebrica:

$$\begin{array}{r} 11001101 + \\ 10011110 = \\ 1\ 01101011 \end{array}$$


Commento (= dire se c'è overflow e spiegare perché nel dettaglio):

- riporto generato in posizione 6: NO
- riporto generato in posizione 7: SI

Poiché i riporti generati nei bit di posizione 6 e 7 sono diversi ho un caso di overflow

NB: la cosa è evidente anche dal fatto che dalla somma di due numeri negativi ottengo un numero positivo. In effetti con 8 bit posso esprimere in complemento a due i numeri da -128 a $+127$ [nella pratica da -127 a $+127$], mentre si ha $-51 - 98 = -149$.

ERRORE TIPICO:

Dire che c'è overflow perché “i bit di posiz. più significativa sono diversi”. In questo caso è vero, ma non c'entra nulla con l'overflow!

Esercizio (Appello dell'11 set 2003)

Rappresentare i numeri -54 e -44 (in base 10) in notazione binaria in complemento a due con 8 bit. Eseguire la somma algebrica dei numeri così ottenuti e commentare il risultato. [3]

Soluzione

- Conversione nella notazione binaria dei numeri (valori assoluti)

$$54 = 00110110_2$$

(32+16+4+2)

$$44 = 00101100_2$$

(32+8+4)

- Rappresentazione in complemento a due dei numeri (con 8 bit)

$$-54 = 11001010$$

$$-44 = 11010100$$

- Somma algebrica:

$$\begin{array}{r}
 11001010 + \\
 11010100 = \\
 \hline
 1\ 10011110 \\
 \swarrow \searrow
 \end{array}$$

e quindi il risultato è 10011110.

Commento:

Non c'è overflow perché i riporti generati nelle posizioni 6 e 7 sono uguali (1 e 1). In effetti risulta un numero negativo pari a $-01100010_2 = -98_{10}$, che è proprio $-54 - 44$

ERRORE COMMESSO DA ALCUNI

Dimenticarsi del numero di bit (8):

$$54 = 110110_2 \quad -54 = 001010$$

$$44 = 101100_2 \quad -44 = 010100$$

e sommando:

$$001010+$$

$$010100=$$

$$011110$$

risultato che non ha alcun senso!!!

(NB: in questo caso l'esercizio è valutato 0 punti!)

Esercizio (Appello del 9 dic 2002)

Rappresentare i numeri 96 e 69 (in base 10) in notazione binaria in complemento a due con 8 bit. Eseguire la somma algebrica dei numeri così ottenuti e commentare il risultato. [4]

Soluzione

Rappresentazione in complemento a due con 8 bit:

$$96 = 01100000 \qquad 69 = 01000101$$

Somma algebrica:

$$\begin{array}{r} 01100000+ \\ 01000101= \\ 10100101 \\ \swarrow \checkmark \end{array}$$

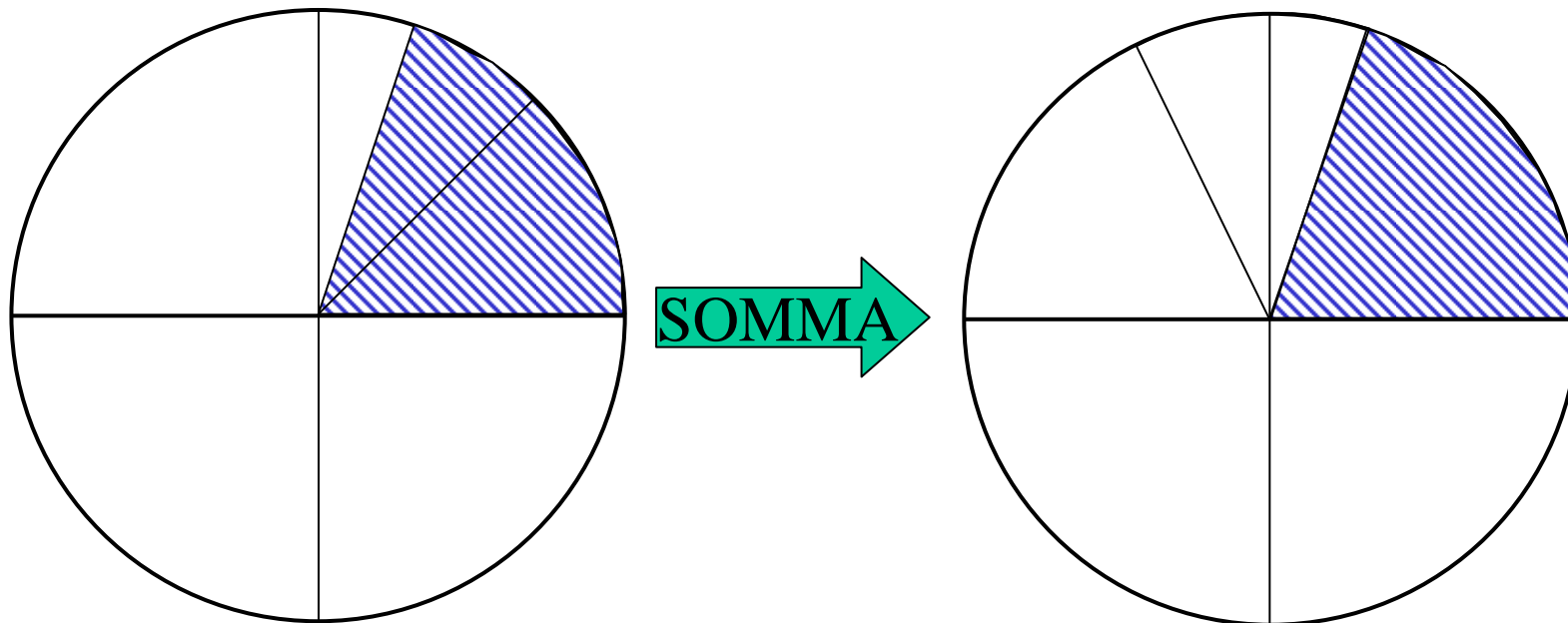
Commento:

Ho overflow perchè i riporti sono diversi (ho soltanto il riporto nel bit 7), cosa deducibile anche dal fatto che il risultato ha il “bit di segno” pari a 1 (rappresenta un numero negativo ottenuto dalla somma di due positivi!).

ERRORE TIPICO: complementare i due numeri che in realtà sono positivi (NB: in questo caso l'esercizio è valutato 0 punti).

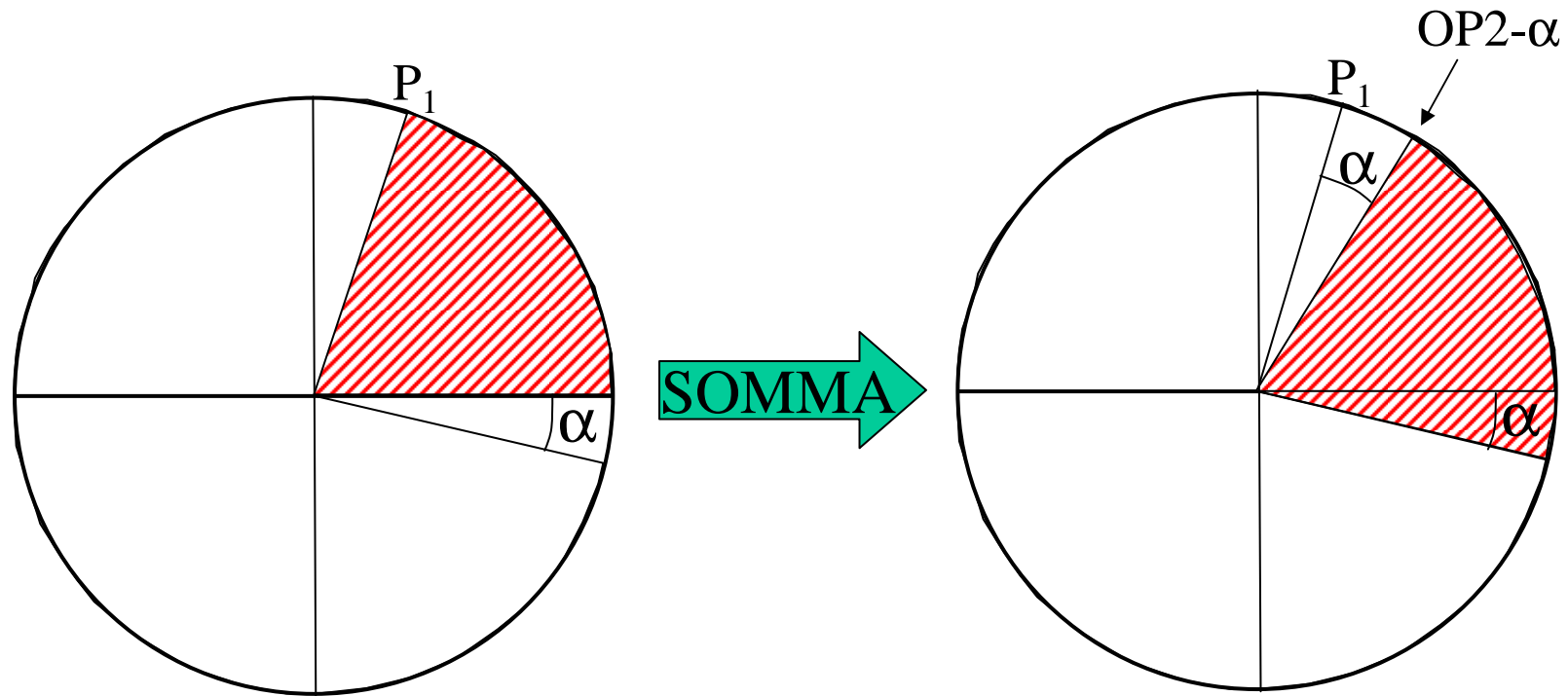
Perché l'operazione di addizione è corretta?

- Esaminiamo intuitivamente la ragione fondamentale.
Vediamo i *tre casi* possibili sul segno degli operandi (OP1 e OP2):
- OP1 (in blu) e OP2 (in rosso) positivi:
usuale operazione di somma tra naturali!



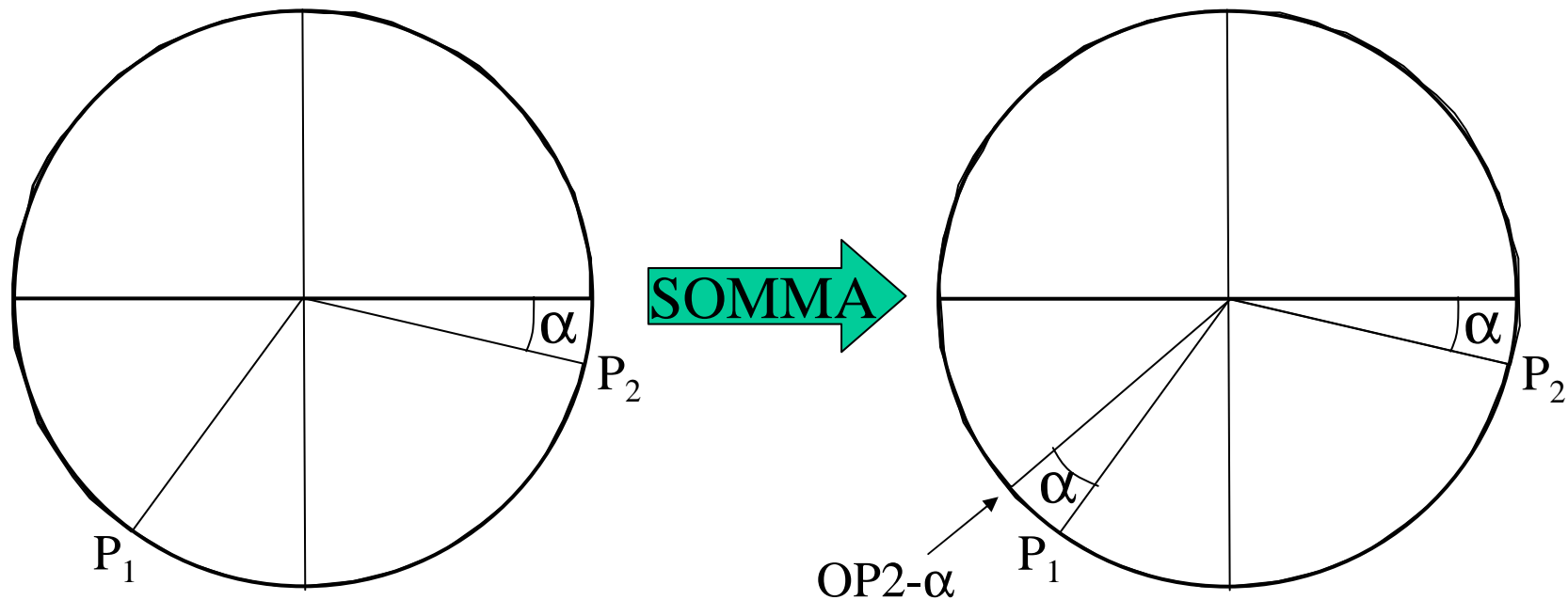
Overflow se il risultato è negativo (semicerchio inferiore)

- OP1 (in blu: $-\alpha$) negativo, OP2 (in rosso) positivo:



Si vede che il risultato è sempre corretto, ovvero non c'è mai overflow

- OP1 (in blu: $-\alpha$) negativo, OP2 (in rosso) negativo:



Si vede che c'è overflow se il risultato è positivo
(semicerchio superiore)

Esercizi proposti

- Dati i seguenti *numeri decimali interi positivi*:
 - 55, 121, 16, 42
- 1) Rappresentarli come *numeri binari su 8 bit*
- 2) Determinare i *numeri negativi corrispondenti in binario* con le seguenti rappresentazioni:
 - *Valore assoluto e segno*
 - *In complemento a 2*

- Fare la *somma* dei numeri binari in complemento a 2 codificati su $n = 8$ bit che corrispondono ai numeri 16_{dieci} e -42_{dieci}
- Fare la *somma* dei numeri binari in complemento a 2 codificati su $n = 6$ bit che corrispondono ai numeri -5_{dieci} e -28_{dieci}

Software per esercitarsi su somme in complemento a due e conversioni

Calcolatrice in complemento a due a 8 bit. Consente di convertire un numero decimale da/a complemento a due (8 bit), eseguire la somma e segnalare l'eventuale condizione di overflow. Può richiedere java (provare con il proprio browser).

Per procurarselo:

Scaricare dal sito del corso il file complementoadue.zip
Nella directory CalcolatriceCa2
si trova un file da visualizzare con il browser.