



Notazione in complemento a 2 caratteristiche ed esempi

- MSB = 0 → numero positivo
 - ▶ stesso valore che si avrebbe in binario puro: il diverso peso del MSB non ha influenza
- MSB = 1 → numero negativo
 - ▶ il valore si ottiene sommando il contributo negativo del MSB con i contributi positivi degli altri bit
- Esempi
 - ▶ $11010111 \rightarrow -128 + 64 + 16 + 4 + 2 + 1 = -41$
 - ▶ $00110011 \rightarrow 32 + 16 + 2 + 1 = 51$
 - ▶ $10000000 \rightarrow -128 + 0 = -128$
 - ▶ $11111111 \rightarrow -128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = -1$
 - ▶ 00000000 denota il valore 0, cioè 0
 - ▶ $01111111 \rightarrow 64 + 32 + 16 + 8 + 4 + 2 + 1 = 127$



Notazione in complemento a 2: proprietà

- Il massimo intero positivo rappresentabile è di uno inferiore al minimo negativo rappresentabile perché lo 0 viene considerato parte dei positivi.
- Lo zero ha ora un'unica rappresentazione → efficienza, semplicità



Complemento a 2: altri esempi

- Con $k = 8$ la rappresentazione in complemento a 2 di -1 si ottiene rappresentando in binario su 8 bit il valore 255, infatti:
 $Cp_2(-1) = 2^8 - 1 = 256 - 1 = 255 \rightarrow 11111111$
- In generale il valore -1 si rappresenta sempre con tutti i bit uguali a uno, indipendentemente da k .



Notazione in complemento a 2 caratteristiche ed esempi

- Valori opposti, come 17 e -17, hanno rappresentazioni diverse
 $17 \rightarrow 00010001$
 $-17 \rightarrow 11101111$
- Rappresentazioni identiche a meno del MSB denotano valori interi completamente diversi
 $01010101 \rightarrow 85$
 $11010101 \rightarrow -128 + 85 = -43$



Notazione in complemento a 2: intervallo di valori rappresentabili

- MSB=0 → k-1 bit usabili come in binario puro → intervallo da 0 a $2^{k-1}-1$
MSB=1 → stesso intervallo traslato di -2^{k-1} → intervallo da -2^{k-1} a -1
- Complessivamente è rappresentabile l'intervallo da -2^{k-1} a $2^{k-1}-1$
- Esempio:
- k=8 bit → $2^8 = 256$ combinazioni → intervallo da -2^7 a 2^7-1 cioè -128..127
 - ▶ anziché, come in binario puro, 0..255
- 256 combinazioni allocate metà ai positivi e metà ai negativi, anziché tutte ai positivi



Notazione in complemento a 2: proprietà

- Le somme algebriche si possono eseguire con le stesse regole dell'aritmetica binaria
 - ▶ La sottrazione si può eseguire banalmente complementando e sommando: $a-b = a+(-b)$
- È verificata la proprietà $X + (-X) = 0$ (trascurando il riporto)
- Non è necessario alcun circuito particolare per trattare i negativi → semplicità e basso costo.



Rappresentazione in complemento a 2: facilità d'uso

- Il problema principale della rappresentazione in modulo e segno è dato dalla non applicabilità degli algoritmi che funzionano per i numeri interi positivi.
- La rappresentazione in complemento a 2 permette di utilizzare direttamente gli algoritmi dei numeri naturali per eseguire le operazioni.
- In particolare è verificata la proprietà $X + (-X) = 0$ (usando le regole dell'aritmetica binaria senza segno).

$$X + (-X) = X + (2^k - X) = 2^k$$

- Ma 2^k non è rappresentabile con k bit: grazie all'overflow si ottengono k zeri.
- Per questo motivo la rappresentazione in complemento a 2 è molto diffusa.



Rappresentazione in complemento a 2 di un numero negativo dato

- Generalizzando alla rappresentazione su k bit:
 - per i negativi: $Cp_2(-N) = \text{binario}(2^k - N)$
 - per i positivi: $Cp_2(+N) = \text{binario}(N)$
 (con $N > 0$)

- Es, con $k = 8$ bits:
 $Cp_2(-39) = \text{binario}(256 - 39) = \text{binario}(217) \rightarrow 11011001$

- Metodo alternativo: sottrarre da 0.

$$-39 = 0 - (+39)$$

$$\begin{array}{r}
 \textcolor{teal}{-1} \textcolor{teal}{-1} \textcolor{teal}{-1} \textcolor{teal}{-1} \textcolor{teal}{-1} \textcolor{teal}{-1} \textcolor{teal}{-1} \textcolor{teal}{-1} \\
 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ - \\
 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ = \\
 \hline
 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1
 \end{array}$$



Rappresentazione in complemento a 2 di un numero negativo dato

- Altro esempio (8 bit)
- $\text{Cp2}(-47) = 256 - 47 = 209 = 11010001$

$$\begin{array}{rcl}
 47 + & 00101111 + \\
 (-47) = & 11010001 = \\
 0 & (1)00000000 = 0
 \end{array}$$



Rappresentazione in complemento a 2: operazione cambio di segno

- Dato un numero X (pos o neg) in Cp2, come posso cambiare di segno?
- cioè: dato $\text{Cp2}(X)$, voglio trovare $\text{Cp2}(-X)$
- Se X è positivo:
 - ▶ Per definizione $\text{Cp2}(-X) = \text{binario}(2^n - X) = \text{binario}((2^n - 1) - (X - 1))$
 - ▶ ma $(2^n - 1)$ è una sequenza di n bit : 11111....
 - ▶ e la differenza tra n uni e un numero binario $(X - 1)$ più piccolo (rappresentato con n bit) equivale a invertire tutti i bit della rappresentazione di $(X - 1)$ -- (provare)
 - ▶ Dopo di che basta aggiungere 1 per ottenere $2^n - X = \text{Cp2}(-X)$
- Quindi: algoritmo: (1) inverto tutti i bit (2) faccio +1
- funziona anche se X è negativo! (provare)



Rappresentazione in complemento a 2: operazione cambio di segno

- Il procedimento funziona anche al contrario, cioè se si applica ad un numero negativo $(-X)$ invertendo i bit e sommando 1 si ottiene la rappresentazione del numero positivo $(+X)$.
- Il procedimento funziona anche per lo zero: sommando uno a una parola di tutti uni si ottiene 0 (trascurando il riporto).

Architettura degli elaboratori

- 59 -

Aritmetica binaria



Rappresentazione in complemento a 2: operazione cambio di segno

- Esempio (con 8 bits)

0 1 0 1 1 1 0 0 ← rappresentazione in cp2 di +92

→ 1 0 1 0 0 0 1 1 (flip di tutti i bit)

→ 1 0 1 0 1 1 0 0 (somma di +1)

 ← rappresentazione in cp2 di -92

- E ritorno, con lo stesso algoritmo:

1 0 1 0 1 1 0 0

→ 0 1 0 1 0 0 1 1 (flip di tutti i bit)

→ 0 1 0 1 1 1 0 0 (aggiunta di +1)

Architettura degli elaboratori

- 60 -

Aritmetica binaria



Rappresentazione in complemento a 2 di un numero negativo dato

Il metodo più semplice per trovare la $cp(-X)$ di un numero negativo $-X$ (X positivo):

1. Trovare $cp(X)$, cioè binario(X)
...e invertirne il segno, cioè:
2. Invertire tutti i bit di tale rappresentazione,
(«farne il complemento a 2», è questo che dà il nome alla notazione!)
3. Aggiungere 1 al risultato così ottenuto.

- Esempio: determinazione della rappresentazione di -25 (su $k=8$ bit)
 - 1) $25 \rightarrow 00011001$
 - 2) flip di segno $\rightarrow 11100110$
 - 3) incremento di 1 $\rightarrow 11100111$
- Verifica: $-128 + 64 + 32 + 4 + 2 + 1 = -128 + 103 = -25$

Architettura degli elaboratori

- 61 -

Aritmetica binaria



Notazione in complemento a 2 Overflow

- Consideriamo i casi seguenti:

-73	10110111	+73	01001001
-73	10110111	+73	01001001

-146	(1)01101110	+146	10010010
- Il risultato è sbagliato: la prima stringa denota +110, la seconda -110.
- Il motivo è che -146 e +146 sono fuori dall'intervallo rappresentabile con 8 bit (-128 ... 127).
- L'overflow si riconosce dal fatto che *sommando due numeri positivi otteniamo un negativo, o viceversa.*

Architettura degli elaboratori

- 62 -

Aritmetica binaria



Overflow

- Sommando due valori che danno come risultato un valore esterno all'intervallo rappresentabile $[-2^{n-1} \dots 2^{n-1}-1]$ si provoca l'invasione del bit di segno da parte del risultato
- In questo caso si ha overflow: il numero di bit non è sufficiente per la rappresentazione del numero.
- Una regola pratica per vedere se si ha overflow:
 - ▶ operazioni tra numeri di segno diverso sono sempre corrette in quanto non si può uscire dal range
 - ▶ operazioni tra numeri dello stesso segno sono corrette se il risultato mantiene il segno degli addendi.




Ricapitolando: rappresentazione di numeri con segno

- Modo 1: modulo e segno
 - ▶ *idea*: usiamo il MSB per il segno, e il resto per il modulo
 - ▶ *es su 8 bit*: 1 bit di segno, 7 bit di modulo
- Modo 2: complemento a due
 - ▶ *idea*: il MSB contribuisce al numero rappresentato con una quantità negativa
 - ▶ *es su 8 bit*: il MSB vale -128, invece di +128

ora vediamo un 3zo modo...

- Modo 3: offset a $\langle X \rangle$ (dove $\langle X \rangle$ è un numero dato)
 - ▶ *idea*: ogni numero viene interpretato come un naturale (senza segno)... «meno X»
 - ▶ *es su 8 bit*: offset a 4: 010 rappresenta $2 - 4 = -2$



Rappresentazione


Offset a <un dato numero X>

- Con n bit rappresento i numeri da $-X$ a $2^n - 1 - X$ (inclusi)
(invece che da 0 a $2^n - 1$)
- Per rappresentare un numero dato, aggiungo X e rappresento lo quel numero in binario (senza segno)
- Per interpretare una sequenza di bit, trovo il numero binario corrispondente (senza segno) e sottraggo X
- Es: su 3 bit: "offset a 5":

	rappresenta:
000	-5
001	-4
010	-3
011	-2
100	-1
101	0
110	1
111	2
- Es: su 3 bit: "offset a 4":

	rappresenta:
000	-4
001	-3
010	-2
011	-1
100	0
101	1
110	2
111	3

Architettura degli elaboratori
- 66 -
Aritmetica binaria



Offset a <un dato numero X>

- Su n bits, offset a $2^{(n-1)}$ ha una caratteristica interessante:
il MSB indentifica il segno (perché?)
- Es: su 3 bit, $2^{(n-1)} = 4$, e "offset a 4":

	rappresenta:
000	-4
001	-3
010	-2
011	-1
100	0
101	1
110	2
111	3


}

strettamente negativi

}

positivi
- Quindi anche con offset a 8 su 4 bit, offset a 128 su 8 bit, offset a 32K su 16 bit, offset a 2G su 32 bit, etc
- Nota: 0 negativo, 1 positivo (il contrario delle altre due rappresentaz.)

Architettura degli elaboratori
- 67 -
Aritmetica binaria

 **Rappresentazioni a confronto**


- Semplice esempio con soli 3 bit

bits	Naturale (<i>unsigned</i>)	Modulo e segno	Compl. a 2 (<i>CP2</i>)	Offset a 4 (o <i>excess4</i>)
000	0	+0	0	-4
001	1	+1	+1	-3
010	2	+2	+2	-2
011	3	+3	+3	-1
100	4	-0	-4	0
101	5	-1	-3	+1
110	6	-2	-2	+2
111	7	-3	-1	+3

↑
MSB
in tutti e tre i casi,
determina il segno

$2^{(3-1)}$

Architettura degli elaboratori - 68 - Aritmetica binaria

 **Rappresentazioni a confronto**

- Modulo e segno:**
 - ⊗ due rappresentazioni *diverse* per lo 0 ($+0 \neq -0$)
 - ⊗ nelle computazioni, si deve sempre tener conto del segno (tanti casi)
→ *è banale, ma non è adottato da nessuna architettura*
- CP2**
 - 😊 lo zero è rappresentato come tutti bit 0
(pratico! vale per tutti i tipi, così il test sullo zero non dipende dal tipo)
 - 😊 somma / differenza: funziona senza modifiche (eccetto overflow)
(rispetto a senza segno)
 - ⊗ confronti ($<$, $>$): funzionano senza modifiche fra positivi e fra negativi,
ma fanno il contrario fra neg e pos: neg $>$ pos (verificare!)
→ *lo standard per gli interi con segno, in tutte le architetture reali*
- Offset a X** se $X = 2^{(n-1)}$
 - ⊗ lo zero non è rappresentato come tutti 0
 - ⊗ somma / differenza: *quasi*: bisogna invertire il MSB dopo l'operaz
 - 😊 confronti ($<$, $>$): funzionano senza modifiche (verificare!)
→ *tipicamente usato in un caso... che stiamo per vedere*

Architettura degli elaboratori - 69 - Aritmetica binaria



Nei linguaggi di programmazione

- Alcuni linguaggi come il C permettono al programmatore di stabilire:
 - Su quanti bit deve essere rappresentato un intero
 - ▶ short, long, ...
 - Specificare se:
 - ▶ il numero deve essere interpretato come intero
 - ▶ (con segno, in complemento a due)
 - ▶ valori $\in [-2^{(n-1)} .. 2^{(n-1)} - 1]$,
 - oppure se:
 - ▶ il numero deve essere interpretato come naturale
 - ▶ (senza segno, in binario)
 - ▶ valori $\in [0 .. 2^n - 1]$.



Nei linguaggi di programmazione

- Rappresentazioni più usate
 - ▶ 8 bit : **“byte”** (o **“char”**)
 - ▶ 16 bit (2 bytes) : **“short int”** o **“short”**
 - ▶ 32 bit (4 bytes) : **“integer”** o **“int”**
(o **“word”**, in un sistema a 32-bit)
 - ▶ 64 bit (8 bytes) : **“long int”** o **“long”**
(o **“word”**, in un sistema a 64-bit)
- Di default, in complemento a due.
 Se numeri naturali: specificare **“unsigned”** (“senza segno”)
 es: **“unsigned char”**, (8 bit senza segno)
“unsigned int” (32 bit senza segno)

Valori massimi e minimi esprimibili?