Corso di
Fondamenti di Informatica

Lezione 4

Nicola Capuano

Dipartimento di Scienze Aziendali, Management & Innovation Systems

ncapuano@unisa.it

Esercizi (soluzioni)

Convertire in decimale i seguenti numeri binari:

$1100_2 = ?_{10}$

$= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8 + 4 = 12_{10}$

$11111111_2 = ?_{10}$

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

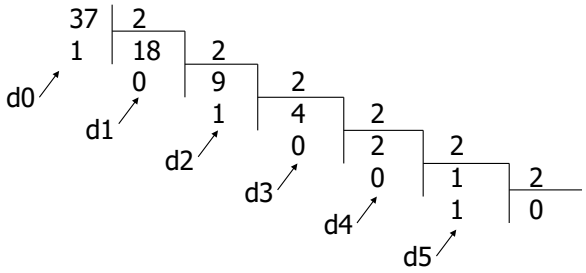
$= 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255_{10}$

Esercizi (soluzioni)

Convertire in binario i seguenti numeri decimali:

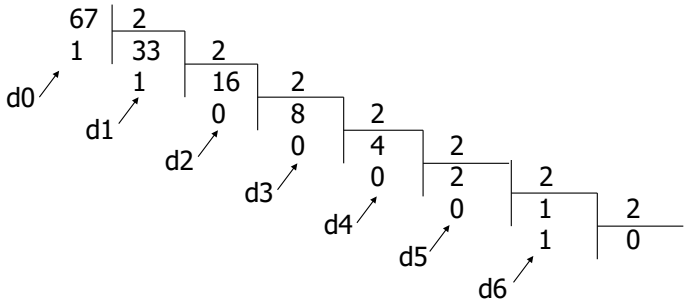
$37_{10} = ?_2$

$= 100101_2$



$67_{10} = ?_2$

$= 1000011_2$



Esercizi (soluzioni)

Eseguire le seguenti operazioni tra numeri binari:

$1010_2 + 11101_2 = ?$

Verifica:

$10_{10} + 29_{10} = 39_{10}$

```
  11
   1010  +
  11101  =
 ------
 100111
```

$10011110_2 + 11011011_2 = ?$

Verifica:

$158_{10} + 219_{10} = 377_{10}$

```
 1  1111
  10011110  +
  11011011  =
 ---------
 101111001
```

Esercizi (soluzioni)

Eseguire le seguenti operazioni tra numeri binari:

$101010_2 - 100_2 = ?$

Verifica:
$42_{10} - 4_{10} = 38_{10}$

```
  02
101010 +
   100 =
------
100110
```

$11011011_2 - 10011110_2 = ?$

Verifica:
$219_{10} - 158_{10} = 61_{10}$

```
   12
  02
    02
     02
 11011011 -
 10011110 =
--------
 00111101
```

Programma del Corso

Modulo 1 - Tecnologie dell'informazione e della comunicazione

- Introduzione alle ICT
- Rappresentazione Digitale dell'Informazione
- Rappresentazione Digitale dei Dati Multimediali
- Architettura Hardware di un Computer
- Software e Sistemi Operativi
- Reti di computer

Rappresentazione Digitale dell'Informazione

Parte 2: Codifica dell'Informazione (continua…)

Bibliografia

- Par 2.4: Il Sistema Binario
- Par. 2.5: Bit e Byte
- Approfondimenti su queste slide

Overflow

In un computer un numero binario è codificato con un numero finito di bit

- Con **n bit** si può rappresentare al massimo il numero $2^n - 1$
- se si supera tale limite si ha un **errore di overflow**

Overflow significa traboccamento

- Il risultato di un'operazione non può essere rappresentato con il numero di bit a disposizione

Esempio: utilizzo una codifica a **8 bit** ed eseguo l'operazione:
11000010 (194) + 01100101 (101) = 100100111 (295)

- Il bit che eccede le capacità di memoria **viene perso** ed **il risultato ottenuto (100111) non è più attendibile**

Overflow

Che risultati si ottengono dalle seguenti operazioni in caso di codifica di un intero con 8 bit?

$00101101_2 + 11101000_2 = ?$

Verifica:

$45_{10} + 232_{10} = 21_{10}$???

Il risultato è errato!

(dovrebbe essere 21+256 = 277)

```
  11  1
  00101101 +
  11101000 =
 100010101
```

bit "perso" — 8 bit

$10011110_2 + 11011011_2 = ?$

Verifica:

$158_{10} + 219_{10} = 121_{10}$???

Il risultato è errato!

(dovrebbe essere 121+256 = 377)

```
 1   1111
  10011110 +
  11011011 =
 101111001
```

bit "perso" — 8 bit

Codifica dei Numeri Relativi

I **numeri interi relativi** sono rappresentati riservando **1 bit per rappresentare il segno**

- Si sceglie il **bit più significativo (quello più a sinistra)** per il segno
- **Se il bit vale 1 allora il segno è –**
- **Se il bit vale 0 allora il segno è +**

Con una codifica a **n bit**, solo **n-1 bit** sono disponibili per rappresentare il valore assoluto del numero

Esistono due codifiche differenti:

- Codifica con **modulo e segno**
- Codifica in **complemento a 2**

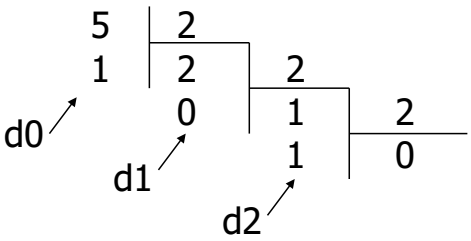
Codifica con Modulo e Segno

Il primo bit rappresenta il segno

Gli altri bit si calcolano con il metodo visto prima

Esempio: -5_{10} = 10000101_2 (codifica con **8 bit**)

- Il **primo bit** (quello più significativo) viene posto a **1** perché il numero è negativo
- gli altri **7 bit** si calcolano con il metodo visto prima applicato al numero 5, ottenendo **0000101**
- Dunque **-5_{10}= 10000101_2**

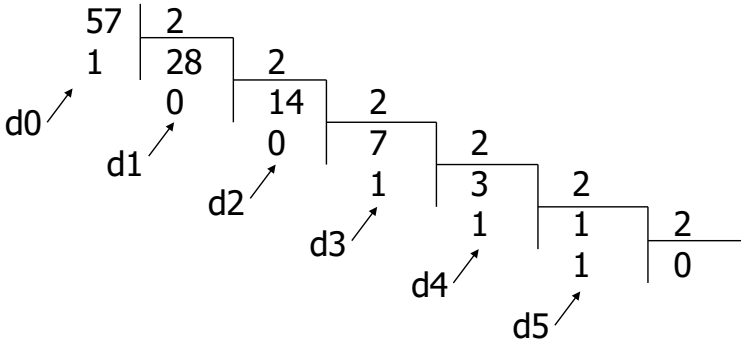


Codifica con Modulo e Segno

Il primo bit rappresenta il segno

Gli altri bit si calcolano con il metodo visto prima

-57_{10} = $?_2$ (codifica con **8 bit**)



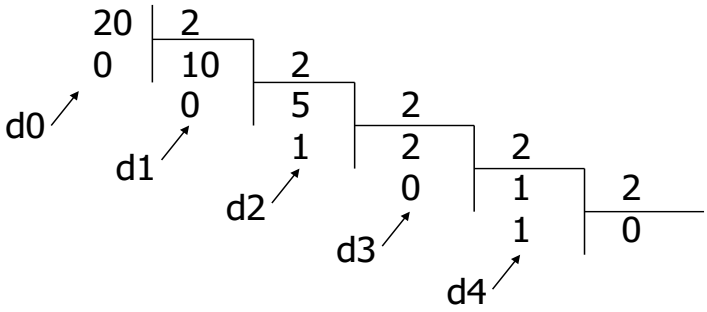
= 10111001_2

Codifica con Modulo e Segno

Il primo bit rappresenta il segno

Gli altri bit si calcolano con il metodo visto prima

$-20_{10} = ?_2$ (codifica con **8 bit**)



$= 10010100_2$

Codifica con Modulo e Segno

Conversione dal binario al decimale:

- moltiplicare ogni cifra del numero binario (tranne la prima) per un'opportuna potenza di 2 e sommare i prodotti
- La prima cifra rappresenta il segno

$10001001_2 = ?_{10}$

Valore assoluto $= 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 +$
$+ 0 \times 2^1 + 1 \times 2^0 =$
$= 1 \times 8 + 1 \times 1 = 8 + 1 = 9_{10}$

Segno: – (la prima cifra è **1**)

Risultato $= -9_{10}$

Codifica con Modulo e Segno

Conversione dal binario al decimale:

- moltiplicare ogni cifra del numero binario (tranne la prima) per un'opportuna potenza di 2 e sommare i prodotti
- La prima cifra rappresenta il segno

$1\,0\,1\,0\,1\,0\,1\,0_2 = ?_{10}$

Valore assoluto $= 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 +$

$+ 1 \times 2^1 + 0 \times 2^0 =$

$= 1 \times 32 + 1 \times 8 + 1 \times 2 = 32 + 8 + 2 = 42_{10}$

Segno: – (la prima cifra è **1**)

Risultato $= -42_{10}$

Codifica in Complemento a 2

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

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

Esempi (codifica con 8 bit):

- il complemento a 2 di 00010100_2 è…
- il complemento a 2 di 01101001_2 è…
- il complemento a 2 di 10000000_2 è…

Codifica in Complemento a 2

i numeri positivi (incluso lo zero) sono rappresentati in modulo e segno (1 bit di segno e n-1 bit per la codifica)

i numeri negativi sono rappresentati con il complemento a 2 della codifica binaria su n bit del valore assoluto

Esempio: codifica di $\mathbf{1_{10}}$ con **8 bit**

- Primo bit = **0** (segno positivo)
- Codifica binaria su 7 bit di 1_{10} = 0000001
- Codifica: $0000000\mathbf{1}_2$

Esempio: codifica di $\mathbf{-1_{10}}$ con **8 bit**

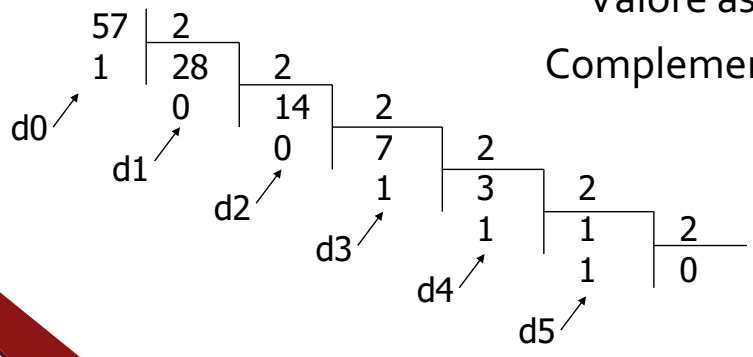
- Codifica binaria del valore assoluto (1) su 8 bits 0000000**1**
- complemento a 2: $\mathbf{11111111}_2$

Codifica in Complemento a 2

i numeri positivi (incluso lo zero) sono rappresentati in modulo e segno (1 bit di segno e n-1 bit per la codifica)

i numeri negativi sono rappresentati con il complemento a 2 della codifica binaria su n bit del valore assoluto

$\mathbf{-57_{10} = ?_2}$ (codifica con **8 bit**)



Valore assoluto = $00\mathbf{111001}_2$

Complemento a 2 = $\mathbf{11000111}_2$

Codifica in Complemento a 2

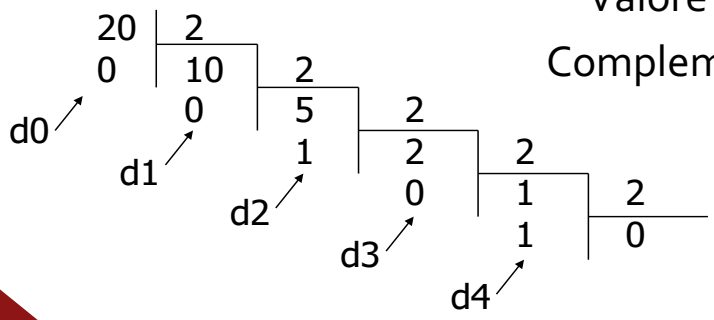
i numeri positivi (incluso lo zero) sono rappresentati in modulo e segno (1 bit di segno e n-1 bit per la codifica)

i numeri negativi sono rappresentati con il complemento a 2 della codifica binaria su n bit del valore assoluto

$-20_{10} = ?_2$ (codifica con **8 bit**)

Valore assoluto = ooo**10100**$_2$

Complemento a 2 = **11101100**$_2$



Codifica in Complemento a 2

Conversione dal binario al decimale:

- Moltiplicare ogni cifra per un'opportuna potenza di 2
- **Sommare i prodotti ottenute dalle n-1 cifre meno significative**
- **Sottrarre il prodotto ottenuto dalla cifra più significativa**

$$00001001_2 = ?_{10}$$

$$= -0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 =$$

$$= 1 \times 8 + 1 \times 1 = 8 + 1 = +9_{10}$$

$$10001001_2 = ?_{10}$$

$$= -1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 =$$

$$= -1 \times 128 + 1 \times 8 + 1 \times 1 = -128 + 9 = -119_{10}$$

Codifica in Complemento a 2

Conversione dal binario al decimale:

- Moltiplicare ogni cifra per un'opportuna potenza di 2
- **Sommare i prodotti ottenute dalle n-1 cifre meno significative**
- **Sottrarre il prodotto ottenuto dalla cifra più significativa**

$01101000_2 = ?_{10}$

$= -0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 +$
$+ 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 =$

$= 1 \times 64 + 1 \times 32 + 1 \times 8 = +104_{10}$

$11101000_2 = ?_{10}$

$= -1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 +$
$+ 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 =$

$= -1 \times 128 + 1 \times 64 + 1 \times 32 + 1 \times 8 = -24_{10}$

Base 10	Codifica valore assoluto (4 bit)	Codifica valore (3 bit)	Codifica in complemento a 2	Codifica in modulo e segno
-8	1000	-	1000	-
-7	0111	-	1001	1111
-6	0110	-	1010	1110
-5	0101	-	1011	1101
-4	0100	-	1100	1100
-3	0011	-	1101	1011
-2	0010	-	1110	1010
-1	0001	-	1111	1001
0	-	000	0000	0000
1	-	001	0001	0001
2	-	010	0010	0010
3	-	011	0011	0011
4	-	100	0100	0100
5	-	101	0101	0101
6	-	110	0110	0110
7	-	111	0111	0111

Comparazione tra Codifiche

Nella notazione in **complemento a 2**, **con n bit**, è possibile rappresentare numeri tra **-2^{n-1} a $2^{n-1}-1$**

- con **4 bit** da **-2^3 (-8)** a **2^3-1 (+7)**
- con **8 bit** da **-128** a **+127**
- con **16 bit** da **-32.768** a **+32.767**
- con **32 bit** da **-2.147.483.648** a **+2.147.483.647**

Nella notazione in **modulo e segno**, **con n bit**, è possibile rappresentare numeri tra **$-2^{n-1}-1$ a $2^{n-1}-1$**

- **con 4 bit da -2^3-1 (-7) a 2^3-1 (+7)**
- **con 8 bit da -127 a +127**
- **con 16 bit da -32.767 a +32.767**
- **con 32 bit da -2.147.483.647 a +2.147.483.647**

La Rappresentazione in Complemento a 2 è più conveniente perché permette un notevole risparmio di tempo nell'esecuzione delle operazioni!

Somma e Differenza

In rappresentazione **modulo e segno**, le operazioni di somma o di differenza dipendono dai segni

Se i segni sono gli stessi:

- Si sommano tutti i bit meno quello del segno
- Si aggiunge il bit di segno

Se i segni sono diversi:

- Si considerano tutti i bit meno quello del segno
- Si sottrae il numero più piccolo in valore assoluto dal più grande
- Si aggiunge il bit di segno del numero in valore assoluto più grande

Troppe regole risultano in calcoli poco efficienti…

Somma e Differenza

In rappresentazione **modulo e segno**, le operazioni di somma o di differenza dipendono dai segni

Se i segni sono gli stessi:

- Si sommano tutti i bit meno quello del segno
- Si aggiunge il bit di segno

$00110010_2 + 00010001_2 = ?$

Verifica:

$+50_{10} +17_{10} = +67_{10}$

```
  11
 00110010 +
 00010001 =
 01000011
```

Somma e Differenza

In rappresentazione **modulo e segno**, le operazioni di somma o di differenza dipendono dai segni

Se i segni sono diversi:

- Si considerano tutti i bit meno quello del segno
- Si sottrae il numero più piccolo in valore assoluto dal più grande
- Si aggiunge il bit di segno del numero in valore assoluto più grande

$10110010_2 + 00010001_2 = ?$

Verifica:

$-50_{10} +17_{10} = -33_{10}$

```
    02
 10110010 –
 00010001 =
 10100001
```

Somma e Differenza

Dati due numeri binari in **complemento a due**, le operazioni di somma e differenza si effettuano **applicano le regole dell'addizione a tutti i bit (compreso il segno)**

- **Il numero binario risultante è già il risultato con il segno giusto.**

11001110_2 + 00010001_2 = ?

```
 11001110 +
 00010001 =
 11011111
```

Verifica:

11001110_2 = -128+64+8+4+2 = **-50_{10}**

00010001_2 = 16+1 = **17_{10}**

11001110_2 = -128+64+16+8+4+2+1 = **-33_{10}**

$-50_{10} + 17_{10} = -33_{10}$

Somma e Differenza

Dati due numeri binari in **complemento a due**, le operazioni di somma e differenza si effettuano **applicano le regole dell'addizione a tutti i bit (compreso il segno)**

- **Il numero binario risultante è già il risultato con il segno giusto.**

00001110_2 + 00011101_2 = ?

```
    111
 00001110 +
 00011101 =
 00101011
```

Verifica:

00001110_2 = 8+4+2 = **14_{10}**

00011101_2 = 16+8+4+1 = **29_{10}**

00101011_2 = 32+8+2+1 = **43_{10}**

$14_{10} + 29_{10} = 43_{10}$

Overflow

Supponiamo di lavorare con codifica in complemento a 2 su **4 bit**

Calcoliamo la somma: $\mathbf{1001_2 + 1111_2}$

Si verifica un overflow

- l'1 più a sinistra viene perso per superamento della capacità dei registri **(limitata a 4 bit!)**
- **Il vero risultato è dunque $\mathbf{1000_2}$**
- **Il risultato è CORRETTO!**

```
 1111
  1001  +
  1111  =
 -----
 11000
```

bit "perso" — 4 bit

Verifica:

$\mathbf{1001_2}$ = -8+1 = $\mathbf{-7_{10}}$
$\mathbf{1111_2}$ = -8+4+2+1 = $\mathbf{-1_{10}}$
$\mathbf{1000_2}$ = $\mathbf{-8_{10}}$

$\mathbf{-7_{10}\ -1_{10} = -8_{10}}$

Overflow

Supponiamo di lavorare con codifica in complemento a 2 su **4 bit**

Calcoliamo la somma: $\mathbf{1001_2 + 1110_2}$

Si verifica un overflow

- l'1 più a sinistra viene perso per superamento della capacità dei registri **(limitata a 4 bit!)**
- **Il vero risultato è dunque $\mathbf{0111_2}$**
- **Il risultato è ERRATO!**

N.B. il vero risultato (-9) non sarebbe rappresentabile con soli 4 bit

- con **4 bit** è possibile rappresentare valori tra $\mathbf{-2^3}$ (**-8**) e $\mathbf{2^3-1}$ (**+7**)

```
  1001  +
  1110  =
 -----
 10111
```

bit "perso" — 4 bit

Verifica:

$\mathbf{1001_2}$ = -8+1 = $\mathbf{-7_{10}}$
$\mathbf{1111_2}$ = -8+4+2 = $\mathbf{-2_{10}}$
$\mathbf{0111_2}$ = 4+2+1 = $\mathbf{7_{10}}$

$\mathbf{-7_{10}\ -2_{10} = +7_{10}}$ **???**

Overflow

Supponiamo di lavorare con codifica in complemento a 2 su **4 bit**

Calcoliamo la somma: $\mathbf{0111_2 + 0001_2}$

Non si verifica alcun overflow

Tuttavia il risultato è ERRATO!

```
 111
 0111 +
 0001 =
------
 1000
```

Verifica:

$\mathbf{0111_2} = 4+2+1 = \mathbf{7_{10}}$
$\mathbf{0001_2} = \mathbf{1_{10}}$
$\mathbf{1000_2} = \mathbf{-8_{10}}$

$\mathbf{7_{10} + 1_{10} = -8_{10}}$ **???**

N.B. il vero risultato (8) non sarebbe rappresentabile con soli 4 bit

- con **4 bit** è possibile rappresentare valori tra $\mathbf{-2^3}$ **(-8)** e $\mathbf{2^3-1}$ **(+7)**

Overflow

Come si fa a capire se il **risultato** di una somma tra numeri rappresentati con codifica in **complemento a 2 è valido?**

Regola:

- se i bit più significativi degli addendi sono uguali tra loro e il bit più significativo della somma è diverso da essi allora il risultato è errato

$\mathbf{1001_2 + 0111_2 = 0000_2}$

Corretto!

$\mathbf{-7_{10} + 7_{10} = 0_{10}}$

```
 1111
 1001 +
 0111 =
------
10000
```

bit "perso" — **4 bit**

I bit più significativi degli addendi sono diversi

Overflow

Come si fa a capire se il **risultato** di una somma tra numeri rappresentati con codifica in **complemento a 2** **è valido?**

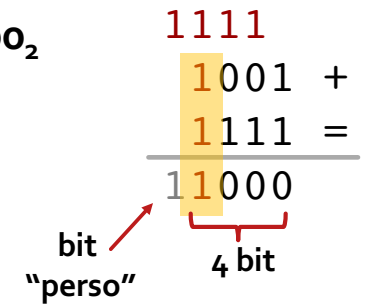
Regola:

- se i bit più significativi degli addendi sono uguali tra loro e il bit più significativo della somma è diverso da essi allora il risultato è errato

$\mathbf{1001_2 + 1111_2 = 1000_2}$

Corretto!

$\mathbf{-7_{10} -1_{10} = -8_{10}}$



I bit più significativi degli addendi sono uguali ma anche quello della somma

Overflow

Come si fa a capire se il **risultato** di una somma tra numeri rappresentati con codifica in **complemento a 2** **è valido?**

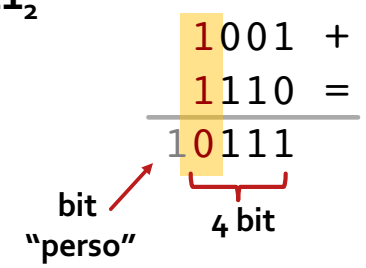
Regola:

- se i bit più significativi degli addendi sono uguali tra loro e il bit più significativo della somma è diverso da essi allora il risultato è errato

$\mathbf{1001_2 + 1110_2 = 0111_2}$

Errato!

$\mathbf{-7_{10} -2_{10} = +7_{10}}$ **???**



I bit più significativi degli addendi sono uguali ma diversi da quello della somma

Overflow

Come si fa a capire se il **risultato** di una somma tra numeri rappresentati con codifica in **complemento a 2 è valido?**

Regola:

- se i bit più significativi degli addendi sono uguali tra loro e il bit più significativo della somma è diverso da essi allora il risultato è errato

$0111_2 + 0001_2 = 1000_2$

Errato!

$-7_{10} + 1_{10} = -8_{10}$???

```
 111
 0111  +
 0001  =
 1000
```

I bit più significativi degli addendi sono uguali ma diversi da quello della somma

Esercizi

Convertire in decimale il seguente numero binario codificato in modulo e segno su 8 bit:

$10010001_2 = ?_{10}$

Valore assoluto: $0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 16 + 1 = 17_{10}$

Segno: –

Risultato: -17_{10}

Convertire in decimale il seguente numero binario codificato in complemento a 2 su 8 bit:

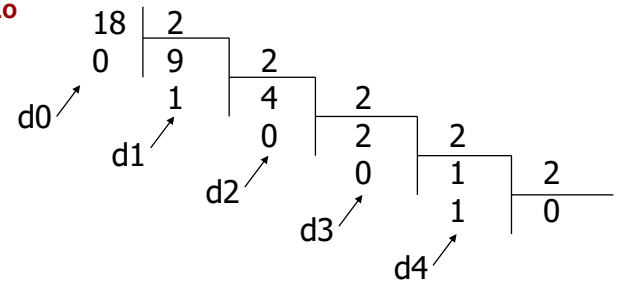
$10010001_2 = ?_{10}$

$= -1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 =$

$= 128 - 16 + 1 = -111_{10}$

Esercizi

Codificare in modulo e segno su 8 bit il seguente numero decimale: -18_{10}



Valore assoluto (su 7 bit) = 0010010

Bit di segno = 1

Risultato = $1\,0010010_2$

Codificare in complemento a 2 su 8 bit il seguente numero decimale: -18_{10}

Valore assoluto (su 8 bit) = $0\,0010010_2$

Complemento a 2 = 11101110_2

Esercizi

Eseguire la seguente operazione tra numeri binari codificati in modulo e segno su 8 bit:

$00010101_2 + 10101110_2 = ?$

```
  02  02
10101110 -
00010101 =
----------
10011001
```

Verifica:

$21_{10} - 46_{10} = -25_{10}$

Eseguire la seguente operazione tra numeri binari codificati in complemento a 2 su 8 bit:

$00010101_2 + 10101110_2 = ?$

```
  1111
00010101 +
10101110 =
----------
11000011
```

Verifica:

$21_{10} - 82_{10} = -61_{10}$

Esercizi per casa

Convertire in decimale i seguenti numero binari codificati in **modulo e segno su 8 bit:**

10110011_2 00100010_2

Convertire in decimale i seguenti numero binari codificati in **complemento a 2 su 8 bit:**

10110010_2 00100010_2

Codificare sia in **modulo e segno** che in **complemento a 2 su 8 bit** i seguenti numeri decimali:

-110_{10} 33_{10}

Eseguire le seguenti operazioni tra numeri binari codificati in **complemento a 2** e verificare la validità del risultato

$10110010_2 + 00100010_2$ $00101111_2 + 00110111_2$

Bibliografia

- Par 2.4: Il Sistema Binario
- Par. 2.5: Bit e Byte
- Approfondimenti su queste slide