Rappresentazione degli interi

Notazione in complemento a 2

- *n* bit per la notazione
  - Nella realta' n=32
  - Per comodita' noi supponiamo **n=4**
- Numeri positivi
  - **0** si rappresenta con 4 zeri **0000**
  - 1 → 0001, 2 → 0010 e cosi' come gia' visto fino al massimo positivo rappresentabile 0111 → **7**
- Numeri negativi
  - -1 si rappresenta con 4 uni **1111** → **-1**
  - -2 -> 1110, -3 → 1101 fino al minimo negativo rappresentabile **1000** → **-8**
- Gli interi rappresentabili con n bit **[-2^{n-1} , 2^{n-1} -1]**
  - Nell'esempio [-2^{4-1},2^{4-1}-1]=[-8,7]

Complemento a due su 3 e 4 bit

a. Using patterns of length three

Bit pattern	Value represented
011	3
010	2
001	1
000	0
111	−1
110	−2
101	−3
100	−4

b. Using patterns of length four

Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

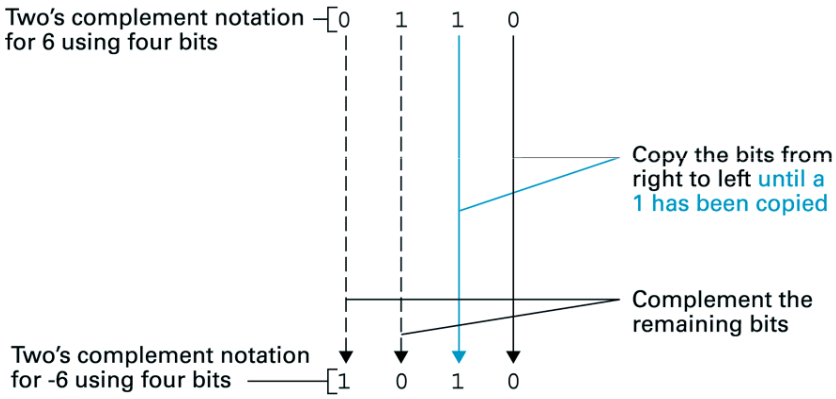
Complemento a due

- **Bit piu' a sinistra: segno** (**0** per positivi, **1** per negativi)
- Confrontiamo **k** e **–k**: da destra a sinistra, uguali fino **al primo 1** incluso, poi una il complemento dell'altra
- Esempio (4 bit): 2=0010, -2=1110

Complemento a due: decodifica

- Se bit di segno =0 ➔ positivo, altrimenti negativo
- Se positivo, basta leggere gli altri bit
- Se negativo, scrivere gli stessi bit da destra a sinistra fino al primo 1, poi complementare, e poi leggere
- Es.: 1010 e' negativo, rappresenta 110 (6), quindi -6

Da k a -k



Metodo alternativo: codifica e decodifica

- **Intero positivo x** ➔ **complemento a due su n bit**:
  se $x \leq 2^{n-1}-1$ scrivo $(x)_2$, altrimenti non e' rappresentabile
  - Esempio: n=4, x=5, $(5)_2=0101$, $x=8>2^3-1=7$
- **Intero negativo –x** ➔ **complemento a due su n bit:**
  se $-x \geq -2^{n-1}$ calcolo $2^n+(-x)=y$ e scrivo $(y)_2$
  - Esempio: n=4, $-x=-3$ $y=2^4-3=16-3=13$ $(13)_2=1101$
- **Compl. a due positivo** (0 = bit + significativo) ➔**decimale**: decodifica dal binario
  - Esempio: n=4, $0111=(7)_2$
- **Compl. a due negativo** (1 = bit + significativo)➔**decimale**: decodifico dal binario a decimale, ottengo y e poi sottraggo $y-2^n$
  - Esempio $1010 = (10)_2$ 10-16=-6

Somma in complemento a due

- Si utilizza il solito metodo
- Anche per sottrazione ➔ basta avere i circuiti per somma e complemento
  - Es. (4 bit): **7-5** = **7 +(-5)** = 0111 + 1011 = 0010
  - 5 = 0101 ➔ -5 = 1011
  - L'eventuale n+1-simo **bit** generato **a sinistra** dal **riporto** deve essere **troncato**
  - Esempio 0111+1011=~~1~~0010

7 -5 2

Esempi di somme

Problem in base ten	Problem in two's complement	Answer in base ten
3 + 2	0011 + 0010 = 0101	5
−3 + −2	1101 + 1110 = 1011	−5
7 + −5	0111 + 1011 = 0010	2

Overflow

- Si **sommano** due numeri positivi tali che il **risultato e' maggiore del massimo numero positivo rappresentabile con i bit fissati** (lo stesso per somma di due negativi)
- Si ha un errore di **overflow** se:
  - Sommando **due positivi** si ottiene **un numero che inizia per 1**: 0101+0100=**1**001, 5+4=-7
  - Sommando **due negativi** viene un **numero che inizia per 0:** 1011+1100= (1)**0**111, -5+(-4)= 7
- Nei computer c'e' overflow con **valori superiori a** 2.147.483.647= 2^{31}

Esercizi

- Da complemento a 2 a base 10:
  - 00011, 01111, 11100, 11010, 00000, 10000
- Da base 10 a complemento a 2 su 8 bit:
  - 6, -6, 13, -1, 0
- Numero piu' grande e piu' piccolo per la notazione in complemento a 2 su 4, 6, 8 bit

Correzioni

- Da complemento a 2 a base 10:
  - **00011→ 3,**
  - 01111→ 15,
  - **11100 → -4,**
  - 11010 → -6,
  - 00000 → 0,
  - 10000 → -16
- Da base 10 a complemento a 2 su 8 bit:
  - 6, -6, 13, -1, 0
  - 00000110, 11111010, 00001101, 11111111, 00000000
- Numero piu' grande e piu' piccolo per la notazione in complemento a 2 su 4, 6, 8 bit
  - **Numero piu' piccolo** -2^{n-1} (n=6 → $-2^5 = -32$)
  - **Numero piu' grande** $2^{n-1} - 1$ (n=6 → $2^5 - 1 = 31$)

Notazione in eccesso

- **n bit** → 2^n possibili **configurazioni binarie** ordinate **da n zeri a n uni**
- Supponiamo per comodita' che **n=4**
- **0** e' rappresentato da un 1 seguito da n-1 zeri:
  0→1000
- n zeri codifica -2^{n-1}: -2^{4-1} = **-8→ 0000** (0-8 = -8)
- n uni codifica $2^{n-1} - 1$: $2^{4-1}-1$= **7 →1111** (15-8 = +7)
- n bit: **notazione in eccesso 2^{n-1}** rispetto al corrispondente binario
  - Es.: 4 bit, notazione in eccesso 8

Notazione in eccesso 8

Bit pattern	Value represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	−1
0110	−2
0101	−3
0100	−4
0011	−5
0010	−6
0001	−7
0000	−8

Esercizi

- Da eccesso 8 a decimale:
  - 1110, 0111, 1000,0010, 0000, 1001
- Da decimale a eccesso 8
  - 5, -5, 3, 0, 7, -8
- Numero piu' grande e piu' piccolo per la *notazione in eccesso* 8, 16, 32

Correzioni (1)

- Da eccesso 8 a decimale:
  - **1110** → 14-8=6
  - **0111** → 7-8=-1
  - **1000, 0010, 0000, 1001**
    0, -6, -8, 1
- Da decimale a eccesso 8
  - **5** → 5+8 → 13 →1101
  - **-5** → -5+8 →3 → 0011
  - **3, 0, 7, -8**
    1011, 1000, 1111, 0000

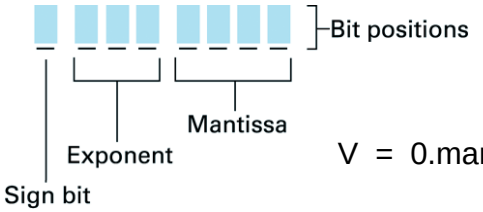
Correzioni (2)

- Numero piu' grande e piu' piccolo per la notazione in eccesso 8, 16, 32
  - **eccesso 8:** $8=2^{n-1}$ → **n=4**
    numero piu' piccolo: **-8**, numero piu' grande **7**
  - **eccesso 16:** $16=2^{n-1}$ → **n=5**
    numero piu' piccolo: **-16** numero piu' grande **15**
  - **eccesso 32:** $32=2^{n-1}$ → **n=6**
    numero piu' piccolo: **-32** numero piu' grande **31**

Rappresentazione dei numeri reali
(floating point)

Rappresentazione dei reali in un computer

- Bisogna rappresentare la posizione della virgola
- **Notazione in virgola mobile (floating point)**: suddivisione in **tre campi**
- Esempio con **8 bit:**
  - Partendo da sinistra: primo bit → **segno** (**0 pos.**, **1 neg.**)
  - Tre bit per **esponente**
  - Quattro bit per **mantissa**



V = 0.mantissa * 2^{exp}

Da floating point a decimale

01101011

1. **Segno:** 0 → positivo, 1→ negativo
2. **Anteporre 0**, alla **mantissa**
   01101011 → 0,1011
3. Interpretare l' **esponente come un numero in eccesso su tre bit** (eccesso 4)
   110→6, 6-4 =2
4. **Spostare la virgola** della **mantissa** della quantita' ottenuta dall'**esponente** a **dx** se il numero positivo a **sx** se e' negativo
   0,1011 → 10,11
5. Tradurre **da binario a decimale** mettendo il segno a seconda del **bit piu' significativo** del foating point
   10,11→ 2,75
6. Aggiungere il segno: **+2,75**

Altro esempio di decodifica

10111100

- Segno: 1 ➔ negativo
- Mantissa: 1100 ➔ 0,1100
- Esponente: 011 ➔ -1 in notazione in eccesso 4 ➔ virgola a sinistra di 1 posto ➔ 0,01100 (3/8, infatti 2x2^(-2) + 2x2^(-3))
- Numero decimale: -3/8 = -0,375

Da decimale a floating point

1. Da decimale a binario:
   0.375 (=3/8)➔ **0,011**
2. La mantissa si ottiene dall'**1 piu' a sinistra** completando con zeri i quattro bit
   1100
3. Contare di **quante posizioni si deve spostare la virgola** per passare da **0,mantissa** a **0,011**. Il numero e' **negativo** se la virgola va a sinistra
   1 bit a sinistra ➔ -1
4. Codificare il numero ottenuto in eccesso 4
   -1 +4= 3 ➔ 011
5. Mettere nel bit piu' significativo il bit di segno
   00111100

Errori di troncamento

- Codifichiamo **2** + **5/8**= 2.625 in **8 bit**
- Binario: 10,101
- Mantissa: vorremmo scrivere 10101, ma abbiamo solo 4 bit ➔ 1010, tronco il bit meno significativo
- Esponente: 110 (2)
- Risultato: **01101010**, che rappresenta 2.5 e non 2 + 5/8
  - Infatti: 0,1010 ➔ 110 (2) ➔10,10 ➔ 2+ ½ = 2.5

Esercizi

- Decodifica: 01001010, 01101101, 00111001
- Codifica: 2.75, 5.25
- Qual e' il piu' grande tra 01001001 e 00111101?

Correzioni (1)

Decodifica:

- 0 100 1010 →5/8 = 0.625 Infatti:
  0 100 1010 --> positivo
  0,1010
  100 --> 4-4=0
  0.1010
  1/2+1/8= 5/8 = 0.625 --> 0.625

- Codifica:
  2.75 --> 0 110 1011 Infatti:
  binario 10,11
  1011 --> 2 posti a dx
  2 --> 110
  0 110 1011

Correzioni (2)

Decodifica:

- 0 110 1101→ 3 + 1/4 = 13/4 = 3.25
- 0 011 1001→ 9/32

Codifica:

- 5.25 → 0 111 1010

Qual e' il piu' grande tra 01001001 e 00111101?

- Il primo e' 0.56, il secondo e' 0.40 → il piu' grande e' il primo