Conversione binario-ottale/esadecimale

- Nella rappresentazione ottale (B=8) si usano gli 8 simboli
  0, 1, 2, 3, 4, 5, 6, 7
- In quella esadecimale (B=16) i 16 simboli
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Con 3 bit si rappresentano $2^3 = 8$ simboli
- Con 4 bit si rappresentano $2^4 = 16$ simboli
- Dato un numero binario, la corrispondente codifica ottale si ottiene considerando gruppi di 3 bit (a partire dalla cifra meno significativa) e sostituendo al valore rappresentato da tali bit la cifra ottale equivalente
- Per convertire in esadecimale si fa la stessa cosa considerando gruppi di 4 bit

Conversione binario-ottale/esadecimale

Esempio:

Dato il numero binario 1100011001011011

la corrispondente codifica ottale è
1/100/011/001/011/011 = 1 4 3 1 3 3

La corrispondente codifica esadecimale è
1100/0110/0101/1011 = C 6 5 B

Rappresentazione di Numeri
Interi Positivi (numeri naturali)

- Un calcolatore assegna un numero fisso N di bit per ogni tipo di dato. N è di solito multiplo di 8.
- Con N bit si possono rappresentare 2^N valori distinti (es. con 8 bit, $2^8=256$), associabili, ad esempio, ai numeri naturali da 0 a $2^N -1$.

Rappresentazione di Numeri Naturali

- **Dati N bit, quali numeri naturali si possono rappresentare?**

I numeri da 0 a $2^N – 1$

- **Dato un numero naturale, quanti bit sono necessari per rappresentarlo ?**

  Per rappresentare un numero naturale I con N bit è necessario che $2^N > I$, cioè $N > \log_2 I$

Es. I = 90 $N > \log_2 90$ $N > 6.492$

Quindi N = 7. Infatti con 7 bit si possono rappresentare i numeri da 0 a 127

Somma

0 + 0 = 0 1 + 0 = 1
0 + 1 = 1 1 + 1 = 10

Somma e sottrazione si eseguono esattamente come nel caso decimale, sulla base dei 4 casi riportati a fianco.

Es. 22 + 21

1		1	← riporto		
	1	0	1	1	0 +
	1	0	1	0	1 =
1	0	1	0	1	1

Rappresentazione di Numeri
Numeri relativi (interi con segno)

- E' possibile estendere in modo naturale la rappresentazione dei numeri naturali ai numeri relativi.
- I numeri relativi possono essere:
  - positivi (segno +)
  - negativi (segno -)
- Il segno può assumere 2 valori

  => Basta aggiungere 1 bit per rappresentare il segno.

Rappresentazione di numeri interi relativi (con N bit)

Prima rappresentazione possibile:

- Segno e Valore Assoluto
  - Il primo bit indica il segno (0=positivo, 1=negativo), cui segue la rappresentazione (su N-1 bit) del valore assoluto del numero, cioè del numero naturale che si ottiene eliminando il segno.

Es. (con 8 bit)

10000011 = - 3

Rappresentazione di numeri interi relativi (con N bit)

Rappresentazione alternativa :

- Complemento a due
  - Un numero negativo **-n** è rappresentato attraverso il **complemento a 2 del suo valore assoluto**, cioè dal numero intero positivo 2^N - n.

Es. N=8 2^N=256

$(6)_{10} = (00000110)_2$

$(-6)_{10} = (11111010)_2$ [Infatti: 256 - 6 = 250]

Osservazioni

- I numeri naturali e i corrispondenti numeri relativi positivi hanno la stessa rappresentazione e hanno come cifra più significativa 0

  $(35)_{10} = (00100011)_2$ in tutte le rappresentazioni

- Tutti i numeri negativi hanno 1 come cifra più significativa (come nella rappresentazione in segno e valore assoluto)

  $(-35)_{10} = (10100011)_2$ rappr. segno + val. ass.

  $(11111010)_2$ rappr. mediante compl. a 2

- Lo zero ha un'unica rappresentazione

- E' possibile realizzare la proprietà dei numeri relativi per cui m – n = m + (-n)

Calcolo rapido del complemento a 2

- Il complemento a 2 di un numero in una rappresentazione ad N bit si definisce come:

  $C(n) = 2^N - n$

- Possiamo scrivere anche $C(n) = 2^N - n - 1 + 1$

N.B. !!

Dati n (ad es. 00100011) ed N (ad es. 8)

la rappresentazione di $2^N - 1 - n$ **è uguale alla rappresentazione di n con le cifre invertite** (11011100)

Allora: **il complemento di un numero si calcola invertendo tutti i suoi bit e sommando 1**

E quindi………..

- Per calcolare il complemento a 2 di un numero
  1. Si rappresenta il numero in binario
  2. Si invertono tutte le cifre (1 ->0 e 0 -> 1)
  3. Si somma 1.

Es.

32 = 00100000

- 32 = 11011111 + 1 = 11100000

N.B.

Il complemento del complemento di un numero è il numero stesso

C (32) = 256 - 32 = 224; (uso 8 bit)

C (C (32)) = 256 - (256 -32) = 32

Esempi di rappresentazioni

- Avendo a disposizione un byte per la rappresentazione, il numero naturale 35 ha la seguente rappresentazione binaria:

  00100011

  Il numero –35 in segno e valore assoluto:

  10100011

  Il numero –35 in complemento a due:

rappr. di 35	00100011
scambio 0<->1	11011100
aggiungo 1	$\boxed{11011101}$

Rappresentazione dello 0

- **modulo e segno**
  - rappresentazione ambigua
  - +0 = 00000000
  - - 0 = 10000000
- **complemento a due**
  - rappresentazione univoca infatti il complemento a due di 00000000 è ancora 0 (*primo vantaggio*!)

Quindi:

in una rappresentazione a N bit con complemento a 2 posso rappresentare i numeri da $-(2^{N-1})$ a $+2^{N-1}-1$

Es. con 8 bit rappresento i numeri da

$-128 = -(2^7)$ a $127 = (2^7-1)$

Addizione

Se si utilizza la notazione con complemento a 2, si può ragionare in termini "algebrici":

Es. 22 – 21

Rappresento -21 in complemento a 2

```
1 11 111 1
0 0 0 1 0 1 1 0
1 1 1 0 1 0 1 1
---------------
0 0 0 0 0 0 0 1
```

La sottrazione equivale alla somma del minuendo con la negazione (rappresentazione complementata) del sottraendo.

Secondo vantaggio della rappresentazione in complemento a 2: **all'interno del calcolatore somme e sottrazioni sono eseguite utilizzando la stessa operazione (circuito).**

Esercizi

- Verificare che, in una rappresentazione senza segno, $(10000101)_2 = (133)_{10}$
- Data una rappresentazione intera a 4 bit senza segno, rappresentare e calcolare in binario le seguenti operazioni (decimali) :
  - 5 + 5
  - 13 + 5 (che risultato "apparente" ottengo ???)
- Data una rappresentazione a 8 bit in complemento a 2
  - determinare il minimo numero rappresentabile
  - determinare il massimo numero rappresentabile

Rappresentazioni a lunghezza fissa: problemi

Usare rappresentazioni di lunghezza fissa porta ad avere valori non rappresentabili (solo un certo numero di configurazioni è disponibile). Questo può provocare errori di due tipi:

Overflow

Underflow

Rappresentazioni a lunghezza fissa: problemi

- **Overflow** indica un errore nella rappresentazione di un certo numero (di solito il risultato di una operazione) dovuto al fatto che la quantità di cifre disponibili è minore rispetto a quelle necessarie a rappresentare il numero.

Es. i due addendi di una somma possono essere rappresentabili ma il risultato della somma no.

Es. (interi senza segno su 8 bit) 130 + 150 =280
Ma con 8 bit rappresento al max. il numero 255.

- **Underflow** indica che il risultato è troppo piccolo per essere rappresentato, cioè minore del più piccolo numero rappresentabile.

Es. divisione fra interi quando il dividendo è minore del divisore.
Il risultato in questo caso è 0.
25/50 = 0.5, ma io posso rappresentare solo 0, 1, …

Esempio 1 (overflow)

- Con 8 bit posso rappresentare:

interi positivi da 0 a 255

interi con segno da -128 a +127

- Supponiamo di essere nel primo caso e di avere 11111111 = $(255)_{10}$

255 + 1 = ?

```
 1 1111111

   11111111+
   00000001
   -------------
 1 00000000
```

- Ma ho solo 8 bit => il risultato dell'operazione è 0!

Esempio 2 (overflow)

- Con 8 bit posso rappresentare:

interi positivi da 0 a 255

interi con segno da -128 a +127

- Supponiamo di essere nel secondo caso e di avere 01111111 = $(127)_{10}$

127 + 1 = ????

```
*1111111*
 01111111+
 00000001
 ---------
 10000000
```

Ma 10000000 è negativo! (-128)

Rappresentazione di Numeri Reali

- Un numero reale è una grandezza continua
- Può assumere infiniti valori
- In una rappresentazione di lunghezza limitata, deve di solito essere approssimato.
- Esistono due forme per rappresentare un numero reale
  - Segno, parte intera, parte decimale (rappresentazione in virgola fissa)
  - Segno, mantissa, esponente (rappresentazione in virgola mobile)

Rappresentazione in virgola fissa (fixed-point)

- Utilizzando una rappresentazione su N cifre si assume che la posizione della virgola sia fissa in un preciso punto all'interno della sequenza. Quindi si assegnano k cifre per la parte intera e N-k cifre per la parte inferiore all'unità (che si può impropriamente definire *parte decimale*)

Es. con 8 cifre e 3 cifre 'decimali'

in base 10	123.45	00123.450
in base 2	111.1	00111.100

Rappresentazione in virgola fissa (fixed-point)

NB In una rappresentazione di tipo posizionale le cifre alla destra della virgola vengono moltiplicate per potenze negative della base

Es.

$(5.75)_{10} = 5*10^0 + 7*10^{-1} + 5*10^{-2}$

$(11.011)_2 = 1*2^1 + 1*2^0 + 0*2^{-1} + 1*2^{-2} + 1*2^{-3}$

Si ricordi che: $N^{-k} = 1 / N^k$

Conversione decimale-binario di numeri non interi

- La parte intera si converte col metodo delle divisioni successive
- Per la parte alla destra della virgola:
  - si moltiplica la parte a destra della virgola per 2
  - si prende la parte intera del risultato come cifra utile
  - si ripetono i 2 passi precedenti finché la parte a destra della virgola non diventa 0

Es. 19.375 Sappiamo che $(19)_{10} = (10011)_2$

0.375 * 2 = **0**.75

0.75 * 2 = **1**.5

0.5 * 2 = **1**.0

Quindi $(19.375)_{10} = (10011.011)_2$