La codifica dell'informazione

Fondamenti di Informatica A

Ingegneria Gestionale

Università degli Studi di Brescia

Docente: Prof. Alfonso Gerevini

Informatica, Informazione e Telecomunicazioni

- l'**informatica** è la *scienza della rappresentazione e dell'elaborazione dell'informazione*
- "l'**informatica** è lo *studio sistematico degli algoritmi che descrivono e trasformano l'informazione: la loro teoria, analisi, progetto, efficienza, realizzazione, applicazione*" [ACM – Association for Computing Machinery]
- le **telecomunicazioni** sono finalizzate "*alla trasmissione rapida a distanza dell'informazione*"

Il concetto di informazione

L'informazione e … il suo supporto fisico

Un brano musicale

Il CD-Rom su cui è memorizzato

Supporto e informazione

- ***Supporto***: mezzo su cui l'informazione può essere ***memorizzata*** e attraverso cui può essere ***trasmessa***
- Non può esistere informazione senza supporto fisico
- Informazione ➔ ***Entità logiche (astratta)***
- Supporto ➔ ***Entità fisiche***
- ***L'informazione si può creare e distruggere***

Proprietà di un supporto

- Il **supporto** deve poter assumere ***configurazioni differenti*** altrimenti non è in grado di portare informazione
- Ad ogni configurazione viene associata una differente ***entità di informazione***
- Il caso più semplice: *2 configurazioni possibili*
- Esempi: interruttore ***acceso/spento***, tensione ***sì/no***, circuito ***aperto/chiuso***
- Alfabeto di 2 simboli, 0 e 1, detti cifre binari o **BIT (*Binary digIT*)**

Codice

Successione di simboli → (Attività di interpretazione) → ***Entità di informazione***

E' necessario un ***codice***:

un insieme di regole che stabiliscono le associazioni fra configurazioni e entità di informazione

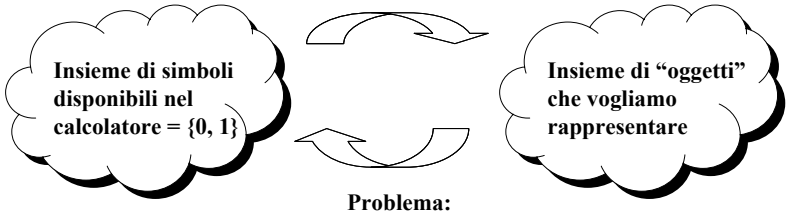
Codifica dei dati e delle istruzioni

- **Programma** = istruzioni che operano su dati
- Istruzioni e dati devono essere rappresentate (**codificate**) secondo il linguaggio noto all'esecutore
- L'esecutore deve essere infatti in grado di memorizzare e manipolare istruzioni e dati
- Nel caso del calcolatore, **istruzioni e dati vengono codificati come sequenze di 0 e 1**

Codifica binaria

- Poiché il nostro esecutore utilizza **componenti a 2 soli stati**, è in grado di riconoscere solamente sequenze di 0 e 1
- **Alfabeto binario = {0, 1}**
- **Importanza tecnologica:**
  - Dispositivi a due stati (livelli di tensione, magnetizzazione, …)
  - Semplicità di realizzazione – Affidabilità
  - "Tutti" i calcolatori elettronici e i dispositivi magnetici di memorizzazione utilizzano tale corrispondenza

Il problema della rappresentazione



Problema:
assegnare un codice univoco a tutti gli oggetti compresi in un insieme

- Ho 2 simboli ed *n* oggetti da codificare, quanto è la lunghezza *k* delle sequenze?
- Oppure: dispongo di sequenze di lunghezza *k* di simboli 0 e 1, quanto è il numero *n* di oggetti che posso codificare?

Codifica binaria

- Se $k = 1$
  - Posso codificare due oggetti (n=2): al primo assegno il codice '0' e al secondo assegno il codice '1'
- Se $k = 2$
  - Posso codificare n=4 oggetti: 00, 01, 10, 11
- Se k = 3
  - Posso codificare n=8 oggetti: 000, 001, 010, 011, 100, 101, 110, 111
- Qual è la **regola**????
- (Ipotesi implicita: i codici hanno tutti la stessa lunghezza)

$n = 2^k$ $\quad k = \lceil \log_2 n \rceil$

- Se ho a disposizione sequenze di k = 5 bit, quanti elementi posso codificare?
  - $n = 2^5 = 32$ elementi
- Se n = 128, di quanti bit ho bisogno (k) per codificarli tutti?
  - $k = \lceil \log_2 128 \rceil = 7$
- …e se n = 129???
- Allora ho bisogno di 1 bit in più! Ottengo uno "spreco" di configurazioni, perché con 8 bit posso codificare fino a 256 elementi

Esempio: i mesi dell'anno

Gennaio Febbraio
Marzo Aprile
Maggio Giugno
Luglio Agosto
Settembre Ottobre
Novembre Dicembre

1 bit ➔ 2 gruppi

Gennaio Febbraio Marzo Aprile Maggio Giugno **0**	
Luglio Agosto Settembre Ottobre Novembre Dicembre **1**	

2 bit ➔ 4 gruppi

Gennaio Marzo Maggio **00**	Febbraio Aprile Giugno **01**
Luglio **10** Settembre Novembre	Agosto **11** Ottobre Dicembre

3 bit ➔ 8 gruppi

Gennaio **000**	Febbraio **010**
Marzo **001** Maggio	Aprile **011** Giugno
Luglio **100**	Agosto **110**
Settembre **101** Novembre	Ottobre **111** Dicembre

4 bit ➔ 16 gruppi… mancano 4 configurazioni!

Gennaio **0000**	Febbraio **0100**
Marzo **0010**	Aprile **0110**
Maggio **0011**	Giugno **0111**
Luglio **1000**	Agosto **1100**
Settembre **1010**	Ottobre **1110**
Novembre **1011**	Dicembre **1111**

Le basi più comuni

- Se la base è *b*, allora le ***cifre*** che possono essere utilizzate per comporre un numero vanno

  da 0 a b-1

  - Esempio: **b = 10**, cifre possibili: [0,1,2,3,4,5,6,7,8,9]
  - Esempio: **b = 2**, cifre possibili: [0,1]
  - Esempio: **b = 8**, cifre possibili: [0,1,2,3,4,5,6,7]
  - Esempio: **b = 16**, cifre possibili: [0,1,2,3,4,5,6,7,8,9,A, B,C,D,E,F]

Conversione binario ⇒ decimale

- Scriviamo i numeri denotando la base attraverso il pedice: es. $\mathbf{1101_{due}}$
- E' facile convertirlo in un numero decimale facendo:
  $1101_{due} = 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 8_{dieci} + 4_{dieci} + 0_{dieci} + 1_{dieci} = 13_{dieci}$
- Altri esempi:
  $10101_{due} = 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 16 + 4 + 1 = 21_{dieci}$
  $110010_{due} = 1x2^5 + 1x2^4 + 0x2^3 + 0x2^2 + 1x2^1 + 0x2^0 = 32 + 16 + 2 = 50_{dieci}$
  $1000001_{due} = 1x2^6 + 0x2^5 + 0x2^4 + 0x2^3 + 0x2^2 + 0x2^1 + 1x2^0 = 64 + 1 = 65_{dieci}$
- Numeri binari ***pari*** e numeri binari ***dispari*** ???

Domande

- Il numero binario 101001011_{due} è pari o dispari?
- A quale numero decimale corrisponde?

$101001011_{due} = (1x2^8 + 0x2^7 + 1x2^6 + 0x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0)_{dieci} = (256 + 64 + 8 + 2 + 1)_{dieci} = 331_{dieci}$

Esempio di conversione da decimale a binario (1)

- $18_{dieci} = ?_{due}$
- 18 div 2 = 9 & 18 mod 2 = 0
- 9 div 2 = 4 & 9 mod 2 = 1
- 4 div 2 = 2 & 4 mod 2 = 0
- 2 div 2 = 1 & 2 mod 2 = 0
- 1 div 2 = 0 & 1 mod 2 = 1

Si legge dal basso verso l'alto !!!

Risultato = 10010_{due}

Esercizio: riconvertire il risultato in decimale

Esempio di conversione da decimale a binario (2)

- $137_{dieci} = ?_{due}$
- 137 div 2 = 68 & 137 mod 2 = 1
- 68 div 2 = 34 & 68 mod 2 = 0
- 34 div 2 = 17 & 34 mod 2 = 0
- 17 div 2 = 8 & 17 mod 2 = 1
- 8 div 2 = 4 & 8 mod 2 = 0
- 4 div 2 = 2 & 4 mod 2 = 0
- 2 div 2 = 1 & 2 mod 2 = 0
- 1 div 2 = 0 & 1 mod 2 = 1

Risultato = **10001001_{due}**

Esercizio: riconvertire il risultato in decimale

I primi 16 numeri in base 10, 2, 8, e 16

Sistema di numerazione			
decimale	**binario**	**ottale**	**esadecimale**
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Perché le basi 2, 8 e 16?

- La rappresentazione binaria ha ***motivazioni di tipo tecnologico***
- Le rappresentazioni ottali ed esadecimali sono utili per rappresentare sinteticamente i valori binari
- E' facile convertire un numero in base 2 in un numero in base 8 o 16
- Le cifre binarie si possono raggruppare **a 3 a 3** e poi codificare con **numeri ottali**
- Le cifre binarie si possono raggruppare **a 4 a 4** e poi codificare con **numeri esadecimali**

Esercizio (in aula)

- Dato il numero binario 001010110111_{due} convertirlo in un numero ottale e poi in un numero esadecimale
- Convertire il numero ottale in numero decimale
- **Numero ottale**: 001 010 110 111 ➔ 1267_{otto}
- **Numero esadecimale**: 0010 1011 0111 ➔ $2B7_{16}$
- **Numero decimale**: $1267_{otto} = (1x8^3 + 2x8^2 + 6x8^1 + 7x8^0)_{dieci} = (512 + 128 + 48 + 7)_{dieci} = 695_{dieci}$

Esercizio (in aula)

- Se la base considerata è b = 4, quali sono le cifre utilizzate per comporre i numeri?
- [0,1,2,3]
- Convertire il numero $(1320)_{quattro}$ nel corrispondente numero in base 10
- $1320_{quattro} = (1x4^3 + 3x4^2 + 2x4^1 + 0x4^0)_{dieci} = (64 + 48 + 8)_{dieci} = 120_{dieci}$
- Qual è il numero massimo rappresentabile in base 3 con quattro cifre (espresso in base 3)?
- 2222_{tre}

Esercizi

- Convertire in formato ***decimale*** i seguenti numeri ***binari***:
  - 11, 101011, 1100, 111111, 10101010
- Convertire in formato ***decimale*** i seguenti numeri ***ottali:***
  - 12, 23, 345, 333, 560
- Convertire in formato ***decimale*** i seguenti numeri ***esadecimali:***
  - 12, DAB, 15D, FFFF, 51A
- Convertire in ***binario*** i seguenti numeri ***decimali***:
  - 45, 234, 67, 83, 972
- Convertire in ***ottale*** e in ***esadecimale*** i ***numeri binari*** ottenuti dalla conversione dei numeri decimali di cui al punto precedente

Operazioni aritmetiche fra <u>numeri naturali</u>

- Operazioni +, -, *, / su numeri in base 2
- **Metodo 1:**
  1. Trasformo i numeri da base 2 a base 10
  2. Sommo i numeri in base 10
  3. Trasformo il risultato da base 10 a base 2
- **Esempio:** $101_{due} + 10_{due} = ?_{due}$
  1. $101_{due} = 5_{dieci}$ e $10_{due} = 2_{dieci}$
  2. $5_{dieci} + 2_{dieci} = 7_{dieci}$
  3. $7_{dieci} = 111_{due}$

Nota che abbiamo definito anche qui un algoritmo!

Operazioni aritmetiche (cont.)

- Per le operazioni in base 2 valgono comunque le ***stesse regole e proprietà delle operazioni in base* 10**
- **Metodo 2:**
  - Possiamo operare direttamente in base 2, usando ad esempio le regole dell'aritmetica binaria

Numeri in complemento a 1

- Il complemento a 1 di un numero ***x*** rappresentato con *n* bit è dato da $\mathbf{2^n-1-x}$
- Ad esempio, con *n*=4 bit, il complemento a 1 del numero *x*=5 è dato da 2^4-1-5 = 16-1-5 = 10
- Quindi $5_{dieci} = 0101_{due}$ e $-5_{dieci} = 1010_{due}$
- Si ottiene più semplicemente complementando tutti i bit **(ovvero si sostituiscono gli 0 con 1 e gli 1 con 0)**

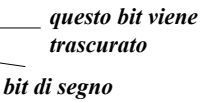
- I **numeri positivi** si rappresentano come nella rappresentazione in valore assoluto e segno
- I **numeri negativi** si rappresentano come complemento a 1 del numero positivo corrispondente

Numeri in complemento a 1

- Ad esempio:
  $n = 8$
  $+21_{dieci} = 00010101_{due}$
  $-21_{dieci} = 11101010_{due}$
- Il numero 0_{dieci} ha due rappresentazioni: 00000000 e 11111111
- Esercizio:
  $n = 8$
  $+36_{dieci} = 00100100_{due}$
  $-36_{dieci} = ????????_{due}$

Numeri in complemento a 2

- La rappresentazione di un numero binario ***x*** in complemento a 2 corrisponde a $2^n+\mathbf{x}$
- *n* è il numero di bit utilizzati per la codifica
- Esempi (con *n* = 4):



  $+6_{dieci} \Rightarrow 2^4 + 6 = 22 \Rightarrow [1]0110 \Rightarrow 0110_{c2}$
  $-6_{dieci} \Rightarrow 2^4 - 6 = 10 \Rightarrow [0]1010 \Rightarrow 1010_{c2}$
  $+3_{dieci} \Rightarrow 2^4 + 3 = 19 \Rightarrow [1]0011 \Rightarrow 0011_{c2}$
  $-3_{dieci} \Rightarrow 2^4 - 3 = 13 \Rightarrow [0]1101 \Rightarrow 1101_{c2}$
- Si possono rappresentare i numeri da $\mathbf{-2^{n-1}}$ **fino a** $\mathbf{2^{n-1}-1}$
- Ad esempio: con *n* = 4 si codificano i numeri da –8 a 7

Codifica in complemento a 2 con 4 bit

Numeri positivi		Numeri negativi	
0000	0	1000	-8
0001	1	1001	-7
0010	2	1010	-6
0011	3	1011	-5
0100	4	1100	-4
0101	5	1101	-3
0110	6	1110	-2
0111	7	1111	-1

Unica rappresentazione del numero zero

Metodi alternativi per il calcolo del complemento a 2

- A partire da ***x*** è possibile calcolare ***–x*** (in complemento a 2) con uno dei seguenti algoritmi:

  Algoritmo 1:

  1. Effettuare il complemento a 1 di ***x***
  2. Aggiungere 1

  Algoritmo 2:

  1. Partendo da destra e andando verso sinistra, lasciare invariati tutti i bit fino al primo 1 compreso
  2. Complementare (invertire) tutti i bit successivi al primo 1

Esempio di uso dell'algoritmo 1

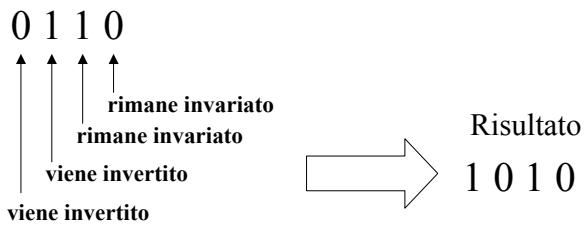
- Dato $+6_{dieci}$ codificato su 4 bit $\Rightarrow$ 0110
- Facendo il complemento a 1 si ottiene 1001
- Sommando 1 al risultato si ottiene…

  1 0 0 1 +
  1 =
  1 0 1 0

- E' corretto? Vedi tabella!

Esempio di uso dell'algoritmo 2

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



Altro esempio

- A partire dalla codifica binaria di 15_{dieci} troviamo la codifica binaria di -15_{dieci}
- 15_{dieci} = 01111 … nota 5 bit!
- ***Usando il primo metodo***:

  01111 $\Rightarrow$ complemento $\Rightarrow$ 10000

  Aggiungo 1 a 10000 $\Rightarrow$ 10001
- ***Usando il secondo metodo:***

  01111 $\Rightarrow$ lascio invariato il primo 1 a destra e complemento tutti gli altri $\Rightarrow$ 10001

Intervalli di rappresentazione

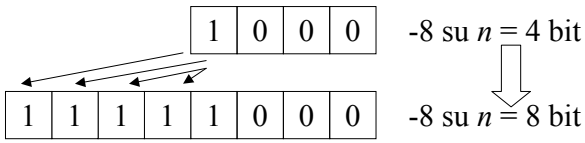
- Supponiamo di avere una codifica con n=16 bit
- ***Rappresentazione in modulo e segno (1 bit per segno, 15 bit per numero in valore assoluto)***: numeri compresi fra –(2^{15}-1) e 2^{15}-1, ovvero fra **–32767 e +32767**… lo 0 ha due rappresentazioni
- ***Rappresentazione in complemento a 1***: numeri 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.

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 **somma** e **sottrazione** possono essere realizzate con un unico circuito: infatti: $x - y = x + (-y)$
- Nel complemento a 2, con n = 16 bit la configurazione **1000000000000000** rappresenta il ***numero negativo più piccolo rappresentabile*** che non ha corrispondenza nei positivi

Estensione del segno

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



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***

n = 8 bit, posso rappresentare i numeri da -2^7 a $+2^7$ - 1

(+5)	00000101	(+5)	00000101
(+8)	00001000	(-8)	11111000
(+13)	00001101	(-3)	11111101

Esempio di addizione

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

```
  0 1 1 0 1 0 +        26
  1 1 0 0 1 1 =        -13        (13 = 0 0 1 1 0 1)
 (1)0 0 1 1 0 1        +13
```

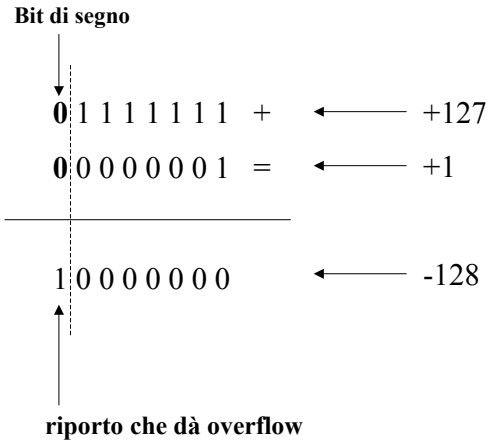
Il riporto viene trascurato

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 (1 bit per il segno e 7 bit per il valore)
- Sommiamo a **01111111 (+127)** il numero **00000001 (+1)** otteniamo un numero negativo **(-128)** invece di **+128**

Esempio di overflow

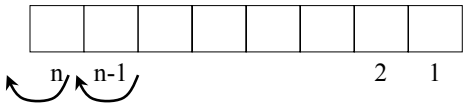


Condizioni di overflow

- Consideriamo solo il caso di numeri binari rappresentati in complemento a 2
- 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**

Regola pratica per l'overflow

- Con una rappresentazione su *n* bit, si ha ***overflow*** se i riporti generati nelle posizioni *n* e *n-1* sono ***diversi***



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 ➔ non è compreso nell'intervallo

1 0 0 1 1 1 +	-25	(25 = 0 1 1 0 0 1)
1 1 0 0 1 1 =	-13	(13 = 0 0 1 1 0 1)
1 0 1 1 0 1 0	+26	

1 0

Esempio di "non" overflow

25 - 13 = 25 + (-13) = 12 ➔ è compreso nell'intervallo

0 1 1 0 0 1 +	25	
1 1 0 0 1 1 =	-13	(13 = 0 0 1 1 0 1)
1 0 0 1 1 0 0	+12	

1 1

Esercizi (1)

- Convertire il numero decimale 15 in rappresentazione binaria complemento a 2 usando 5 bit
- Convertire il numero decimale -35 in rappresentazione binaria complemento a 2 usando 7 bit
- Convertire 01101 da rappresentazione binaria complemento a due nel corrispondente numero decimale
- Invertire il segno dei seguenti numeri binari in rappresentazione complemento a 2
- Indicare l'intervallo di rappresentazione in complemento a due per numeri binari su 8 bit
- Sommare i seguenti numeri binari in rappresentazione Modulo e Segno:

  100011 + 001111 001101 + 111000

  100110 + 111001 011100 + 011100

Esercizi (2)

- La somma dei seguenti numeri binari in rappresentazione complemento a due su 5 bit determina una condizione di overflow?

  01110 + 10011 01011 + 01010 11101 + 10111

- Determinare il risultato delle seguenti addizioni e sottrazioni tra numeri binari in rappresentazione complemento a due su 5 bit:

  01001 + 01111 10111 + 10000

  01000 - 01110 00110 - 10000

Esercizi (3)

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

Esercizi (4)

- 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}