POLITECNICO DI MILANO

La codifica binaria

Informatica B

Daniele Loiacono

Introduzione

- Il calcolatore usa internamente una codifica binaria (0 e 1) per rappresentare:
  - i dati da elaborare
  - le istruzioni dei programmi eseguibili

- Fondamenti di codifica dell'informazione:
  - Codifica dei numeri
    - Naturali
    - Interi
    - Frazionari
  - Codifica dei caratteri
  - Codifica delle immagini
  - Algebra di Boole

Codifica numeri naturali

Rappresentazione in base *p*

- *Metodo posizionale*: ogni cifra ha un *peso* che dipende dalla posizione
  Esempio: 123 = **1**00 +**2**0 +**3**
  312 = **3**00 +**1**0 +**2**
- Di solito noi usiamo la *base* decimale
- Un numero generico di *m* cifre è rappresentato dalla sequenza: $a_n, a_{n-1}, a_{n-2}, \ldots, a_0$

  a_n : cifra più significativa
  a_0 : cifra meno significativa
  $n = m-1$
  $a_i \in \{0, 1, \ldots, p-1\}$

Rappresentazione in base *p*

- Un numero naturale *N*, composto da *m* cifre, in base *p*, si esprime come:

$$N_p = a_n \cdot p^n + a_{n-1} \cdot p^{n-1} + \ldots + a_1 \cdot p^1 + a_0 \cdot p^0 = \sum_{i=0}^{n} a_i \cdot p^i$$

- Esempio in *base decimale* (*p=10*):
  $587_{10} = 5 \cdot 10^2 + 8 \cdot 10^1 + 7 \cdot 10^0$
- Posso rappresentare i numeri nell'intervallo discreto:
  $[0 \ , \ p^m - 1]$

Rappresentazione in base due

- Base binaria: $p=2$; cifre $a_i \in \{0, 1\}$ chiamate *bit* (*binary digit*)
- Otto bit sono chiamati *byte*
- Esempio, con $m=5$:
  $11011_2 = (1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0)_{10} = 27_{10}$
- Posso rappresentare i numeri nell'intervallo discreto:
  $[0 \ , \ 2^m - 1]$
- Esempio con $m=8$:
  rappresento numeri binari: $[00000000_2 \ , \ 11111111_2]$,
  ovvero: $[0 \ , \ 255]$

Conversioni di base

- Per convertire da base due a base 10:
  - Usare la sommatoria illustrata nella slide precedente
- Per convertire da base dieci a base due:
  - Metodo delle divisioni successive
  - Esempio: $13_{10} = 1101_2$
    - 13/2 = 6 resto = 1
    - 6/2 = 3 resto = 0
    - 3/2 = 1 resto = 1
    - 1/2 = 0 resto = 1

1101

Basi ottale ed esadecimale

- *Base ottale*: *p=8*; $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$
  - Esempio: $234_8 = (2 \cdot 8^2 + 3 \cdot 8^1 + 4 \cdot 8^0)_{10} = 156_{10}$
- *Base esadecimale*: *p=16*; $a_i \in \{0, 1, 2, \ldots, 9, A, B, C, D, E, F\}$
  - Esempio: $B7F_{16} = (11 \cdot 16^2 + 7 \cdot 16^1 + 15 \cdot 16^0)_{10} = 2943_{10}$
  - Notare: “11” al posto di “B” e “15” al posto di “F”, i loro equivalenti in base dieci

Somma

- Si somma cifra per cifra
- La somma può generare un riporto
- Il riporto dovrà essere considerato nella somma seguente

Riporto precedente	Somma	Risultato	Riporto
0	0 + 0	0	0
0	0 + 1 1 + 0	1	0
0	1 + 1	0	1
1	0 + 0	1	0
1	0 + 1 1 + 0	0	1
1	1 + 1	1	1

Somma e carry

- Esempio:

```
   1    ← riporto
 0101 +          (5₁₀)
 1001 =          (9₁₀)
 ------
 1110            (14₁₀)

111     ← riporti
 1111 +          (15₁₀)
 1010 =          (10₁₀)
-------
carry → 11001    (25₁₀ se uso 5 bit;
                  9₁₀ se  considero 4 bit: errato)
```

Codifica numeri interi

Modulo e segno

- Occorre codificare anche il “segno”
- Uso un bit per memorizzare il segno: “1” significa numero negativo, “0” numero positivo. Esempio *m=3*:

Num. intero, base 10	Num. intero, base due, modulo e segno
–3	111
–2	110
–1	101
–0	100
+0	000
+1	001
+2	010
+3	011

Complemento a due (CPL_2)

- Usando m bit: $(-N)_{CPL2} = (2^m - N_{10})_2$
- Esempio (m=3): $(-N)_{CPL2} = (2^3 - N_{10})_2$

Num. intero base 10	**Trasformazione**	**Num. intero, base 2, CPL_2, *m=3***
-4	8 - 4 = 4	4_{10} = 100
-3	8 - 3 = 5	5_{10} = 101
-2	8 - 2 = 6	6_{10} = 110
-1	8 - 1 = 7	7_{10} = 111
0	nessuna	0_{10} = 000
1	nessuna	1_{10} = 001
2	nessuna	2_{10} = 010
3	nessuna	3_{10} = 011

Complemento a due (CPL_2)

- Posso rappresentare i numeri nell'intervallo discreto: $[-2^{m-1}\ ,\ 2^{m-1} - 1]$
  - Asimmetria tra negativi e positivi
  - Esempio (*m=8*): [-128, +127], perché $-2^7 = -128$ e $2^7 - 1 = +127$
- Tutti i numeri negativi cominciano con il bit più significativo posto a "1", mentre tutti i positivi e lo zero iniziano con uno "0"
- Codifica da base 10 a complemento a 2
  - Rappresentare $2^m - N$
  - Complemento tutti i bit e sommo 1

Somma e sottrazione in CPL_2

- Somma: come per i naturali
- Sottrazione: $N_1 - N_2 = N_1 + (-N_2)_{CPL2}$
- Carry:
  - Il carry non viene considerato!
- Overflow:
  - Se, sommando due interi di *m* bit dotati di segno concorde, ottengo un risultato di segno discorde (sempre considerando *m* bit), allora si ha un *overflow* (il risultato non è codificabile su *m* bit) e l'operazione è errata
  - L'overflow non può verificarsi se gli operandi sono di segno discorde

Codifica numeri frazionari

Parte frazionaria di un numero

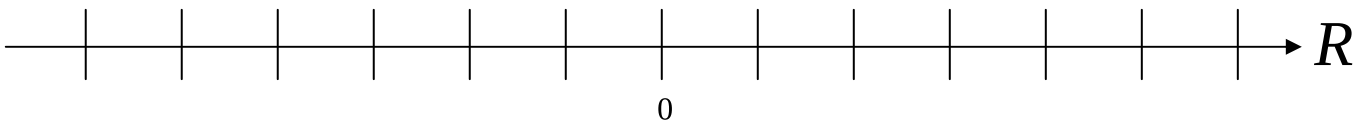
- Rappresentiamo la parte frazionaria di un numero reale
- In base due, un numero frazionario *N*, composto da *n* cifre, si esprime come:

$$N_2 = a_{-1} \cdot 2^{-1} + a_{-2} \cdot 2^{-2} + \ldots + a_{-n} \cdot 2^{-n} = \sum_{i=-n}^{-1} a_i \cdot 2^i$$

- Esempio con $n=3$: $0{,}101_2 = (1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3})_{10} = 0{,}625_{10}$
- Date *n* cifre in base $p=2$, posso rappresentare numeri nell'intervallo continuo: $[0 \ , \ 1-2^{-n}]$
- L'errore di approssimazione sarà minore di 2^{-n}

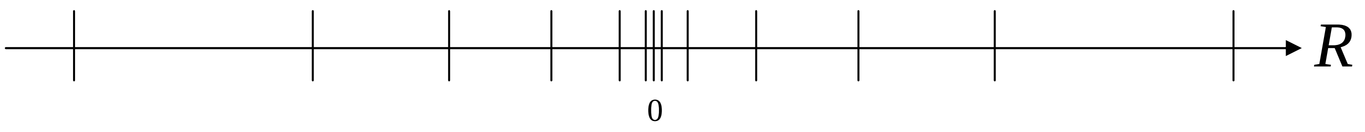
Virgola fissa

- Uso *m* bit e *n* bit per parte intera e frazionaria
  - Esempio (*m=8*, *n=6*, tot. 14 bit): $-123{,}21_{10}$
    $-123_{10} = 10000101_2$
    $0{,}21_{10} \approx 001101_2$
    $-123{,}21_{10} \approx 10000101{,}001101_2$
- Come scelgo *m* e *n*?
- Precisione costante lungo l'asse reale ***R***:



Virgola mobile (floating point)

- Il numero è espresso come: $r = m \cdot b^n$
  - *m* e *n* sono in base *p*
  - *m:* mantissa (numero frazionario con segno)
  - *b:* base della notazione esponenziale (numero naturale)
  - *n:* caratteristica (numero intero)
  - Esempio (p=10, b=10): -331,6875 = $-0{,}3316875 \cdot 10^3$
    m = -0,3316875; n = 3
- Uso l_1 bit e l_2 bit per codificare *m* e *n*
- Precisione variabile lungo l'asse reale ***R***:



Codifica caratteri

Caratteri

- Codifica numerica
- ASCII (American Standard Code for Information Interchange) utilizza 7 bit (estesa a 8 bit)
- L'ASCII codifica:
  - I caratteri alfanumerici (lettere maiuscole e minuscole e numeri), compreso lo *spazio*
  - I simboli (punteggiatura, @, #, …)
  - Alcuni caratteri di controllo che non rappresentano simboli visualizzabili (TAB, LINEFEED, RETURN, BELL, ecc)

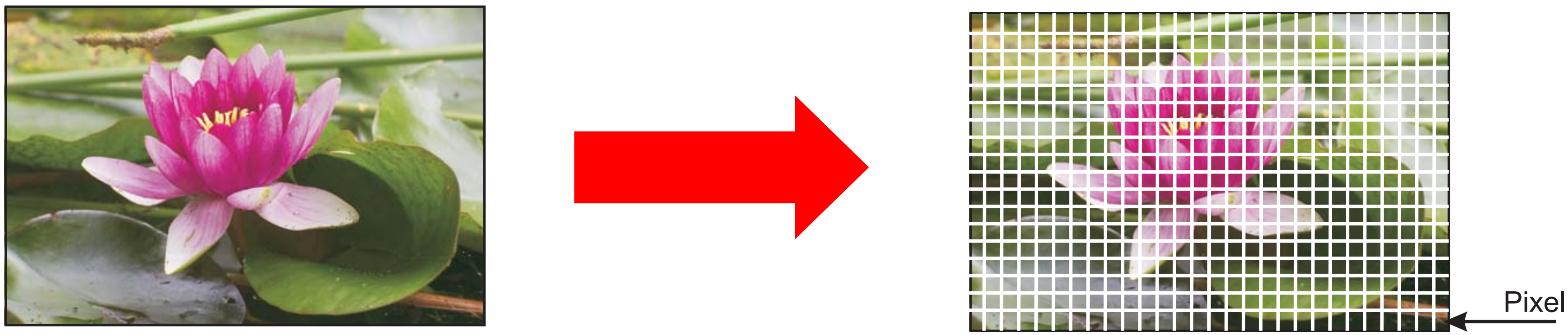
Tabella ASCII (parziale)

DEC	CAR	DEC	CAR	DEC	CAR	DEC	CAR	DEC	CAR
48	0	65	A	75	K	97	a	107	k
49	1	66	B	76	L	98	b	108	l
50	2	67	C	77	M	99	c	109	m
51	3	68	D	78	N	100	d	110	n
52	4	69	E	79	O	101	e	111	o
53	5	70	F	80	P	102	f	112	p
54	6	71	G	81	Q	103	g	113	q
55	7	72	H	82	R	104	h	114	r
56	8	73	I	83	S	105	i	115	s
57	9	74	J	84	T	106	j	116	t
				85	U			117	u
				86	V			118	v
				87	W			119	w
				88	X			120	x
				89	Y			121	y
				90	Z			122	z

Codifica immagini

L’immagine digitale

- Le immagini sono codificate come sequenze di bit
- *Digitalizzazione*: passaggio dall’immagine alla sequenza binaria
- L’immagine è suddivisa in una griglia di punti (detti *pixel*)
- Ogni pixel è descritto da un numero (su 8, 16, 24, o 32 bit) che ne rappresenta il colore (es. con 8 bit → $2^8 = 256$ combinazioni di colore)
- *Dimensioni* dell’immagine: larghezza e altezza, in pollici



L'immagine digitale

- *Risoluzione*: è data come numero di pixel per pollice (dpi - *dot per inch*)
  - Spesso (ma non sempre) la risoluzione orizzontale è uguale a quella verticale
- Standard di codifica:
  - TIFF, PNG: comprimono l'immagine, per ridurne l'occupazione, senza deteriorarla (compressione *lossless*)
  - JPEG: comprime (molto di più), ma deteriora l'immagine (compressione *lossy*)

Algebra di Boole

Algebra di Boole

- E’ basata su tre operatori: AND, OR, NOT
- Ogni formula può assumere solo due valori: “vero” o “falso”. Idem per le variabili
- Rappresentiamo “vero” con “1” e “falso” con “0”
- AND e OR sono operatori *binari*
- NOT è un operatore *unario*

Operatori booleani

- Tavole di verità:

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

A	NOT A
0	1
1	0

Operatori booleani: proprietà

- Commutativa:
  - A OR B = B OR A
  - A AND B = B AND A
- Distributiva di uno verso l’altro:
  - A OR (B AND C) = (A OR B) AND (A OR C)
  - A AND (B OR C) = (A AND B) OR (A AND C)
- Leggi di De Morgan:
  - A AND B = NOT ((NOT A) OR (NOT B))
  - A OR B = NOT ((NOT A) AND (NOT B))

Espressioni booleane

- Regole di precedenza:
  - NOT ha la massima precedenza
  - poi segue AND
  - infine OR
- Se voglio alterare queste precedenze devo usare le parentesi (a volte usate solo per maggior chiarezza)
- Per valutare un espressione booleana si usa la *tabella della verità*
- Due espressioni booleane sono uguali se e solo se le tabelle della verità sono identiche

Dalla formula alla tabella

- Vediamo un esempio, per l'espressione:
  D = A AND NOT (B OR C)

A	B	C	D = A AND NOT (B OR C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0