POLITECNICO DI MILANO

Codifica dell'informazione

Informatica B

Daniele Loiacono

Come memorizzo l'informazione nel calcolatore?

1 bit di informazione…

1 bit di informazione…

La memoria del calcolatore

L'informazione nel calcolatore…

- Il calcolatore contiene *elementi di memoria* che possono avere solo due stati (0 o 1) per rappresentare:
  - i dati da elaborare (numeri, testi, immagini, suoni,…)
  - le istruzioni dei programmi eseguibili
- Le informazioni:
  - In input vengono codificate
  - In output vengono decodificate
- 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 = 100 +20 +3
  312 = 300 +10 +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},..., a_0

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

Rappresentazione in base *p*

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

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

$$\forall i \qquad 0 \le a_i \le p-1$$

- 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 $m=5$:
  $11011_2 = (1\cdot2^4+1\cdot2^3+0\cdot2^2+1\cdot2^1+1\cdot2^0)_{10} = 27_{10}$
- 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 +          (5_10)
 1001 =          (9_10)
 ------
 1110            (14_10)
```

```
111     ← riporti
 1111 +          (15_10)
 1010 =          (10_10)
-------
11001            (25_10 se uso 5 bit;
                  9_10 se considero 4 bit: errato)
```

carry →

Esercizi:

- Scrivere tutti i numeri binari (e il valore decimale) che possono essere rappresentati con 4 bit
- Convertire i numeri 5 e 9 in base 2 usando 4 bit, e eseguire la somma
- Convertire i numeri 41 e 18 in base 2 usando 6 bit, e eseguire la somma
- Eseguire la somma dei numeri binari: 01011 e 00110
- Eseguire la somma dei numeri binari: 01111 e 10110

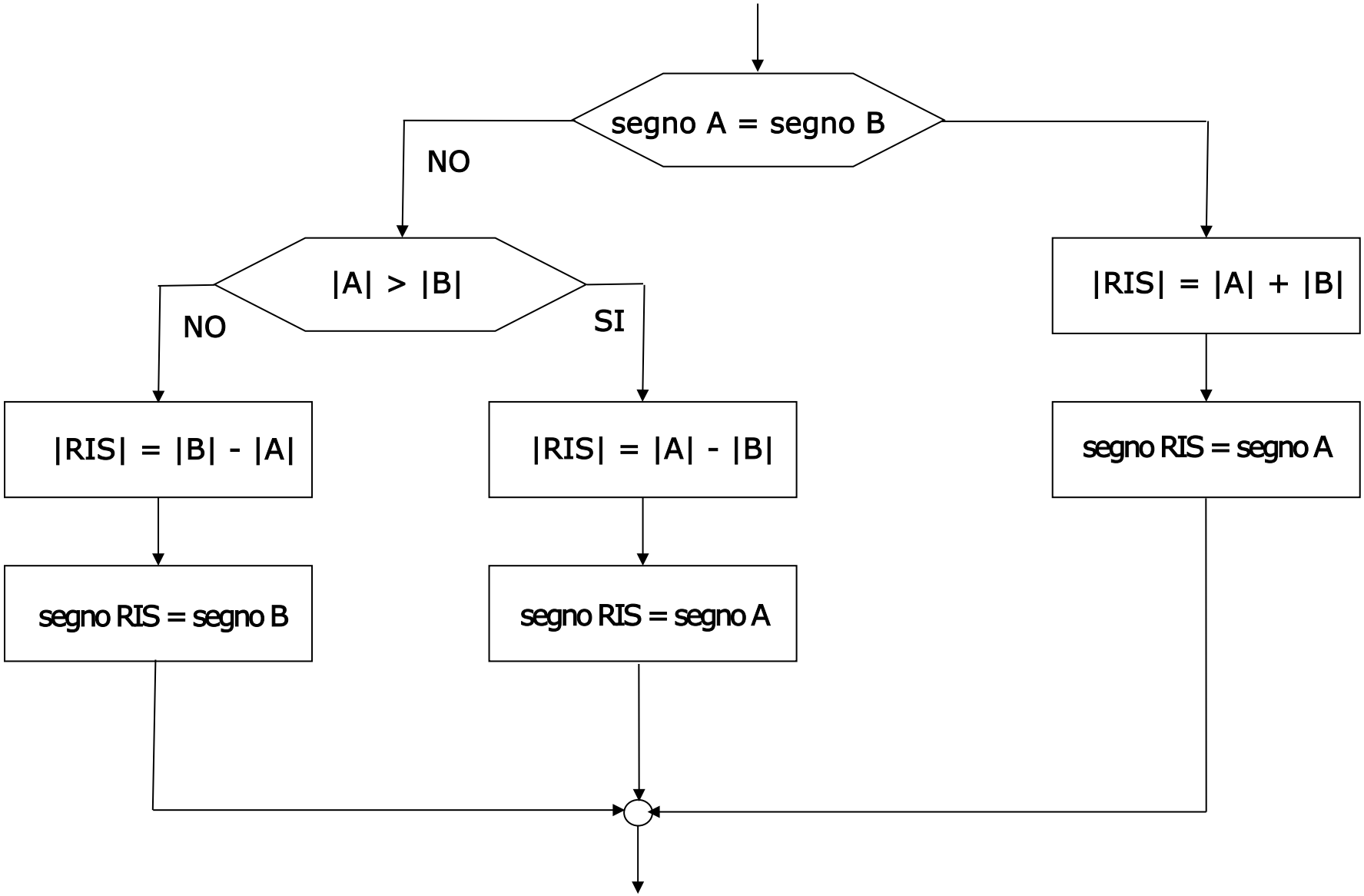
Codifica numeri interi

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

Problemi codifica modulo e segno



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 di –N da base 10 a complemento a 2
  - Rappresentare $2^m - N$
  - Rappresento 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

Esercizi:

- Elencare tutti i numeri binari in CPL_2 di 4 bit
- Convertire i numeri 11 e –7 in CPL_2 e effettuarne la somma
- Eseguire in binario, 5 – 13 e convertire il risultato in decimale
- Eseguire in binario (4 bit), 5 - 5 : c'è overflow?
- Eseguire in binario (4 bit), 5 + 5 : c'è overflow?

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:

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

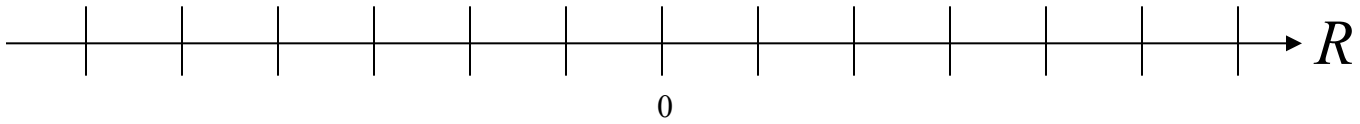
- Esempio con *n=3:* $0{,}101_2 = (1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3})_{10} = 0{,}625_{10}$
- Date *n* cifre in base *p=2*, posso rappresentare numeri nell'intervallo continuo: $[0 \ , \ 1\text{-}2^{-n}]$
- L'errore di approssimazione sarà minore di 2^{-n}

Esercizi

- Convertire in binario: 0,125
- Convertire in binario: 0,375
- Convertire in decimale: 11,11
- Elencare tutti i numeri con solo 3 bit per la parte frazionaria

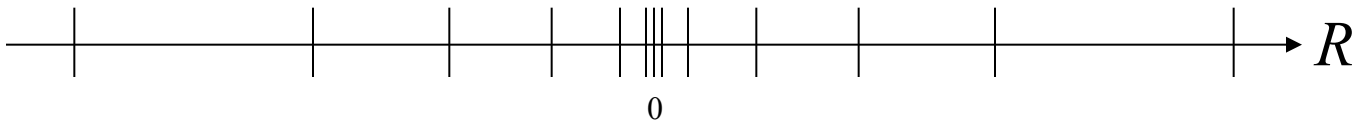
Virgola fissa

- 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***:



Standard IEEE 754-1985

- Il numero è espresso come: $[S]M \cdot 2^n$

- 1 bit per il segno S
- Mantissa M normalizzata tra 1.0000.. e 1.11111…
- La parte intera (sempre 1) della mantissa viene omessa
- L'esponente viene memorizzato in eccesso K
  - E = n + K
  - K = $2^{m-1} - 1$ *(se m=8 K=127)*

Standard IEEE 754-1985

Campo	Precisione singola	Precisione doppia	Precisione quadrupla
Ampiezza tot	32	64	128
S	1	1	1
E	8	11	15
M	23	52	111
K	127	1023	16383

Esercizi

- Codificare secondo lo standard IEEE a precisione singola il seguente numero decimale: 42.6875
- Codificare secondo lo standard IEEE a precisione singola il seguente numero decimale: 0.875
- Convertire in base dieci il seguente numero espresso nella codifica floating point:
  - S=0
  - M=10010011 0000000 0000000
  - E=10000100

Soluzioni

1. X = 42.6875 -> 101010.1011 = 1.010101011 × 2^5
   - S = 0 (1 bit)
   - E = 5 + K = 5 + 127 = 132 -> 10000100 (8 bit)
   - M = 01010101 10000000 0000000 (23 bit)
2. X = 0.875 -> 0.111 = 1.11 × 2^-1
   - S = 0 (1 bit)
   - E = -1 + K = -1 + 127 = 126 -> 01111110 (8 bit)
   - M = 11000000 00000000 0000000 (23 bit)
3. E=10000100 → E = 128+4 – 127 = 5
   1.M (1.10010011 0000000 0000000) → 110010.011
   110010 = 32+16+2 = 50
   .011 = 0.25+0.125 = 0.375
   → X=50.375

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*)

Codifica di un programma

Forma binaria di un programma

0100000000010000	**Leggi** un valore dall'input e mettilo nella cella 16 (**a**)
0100000000010001	**Leggi** un valore dall'input e mettilo nella cella 17 (**b**)
0100000000010010	**Leggi** un valore dall'input e mettilo nella cella 18 (**c**)
0100000000010011	**Leggi** un valore dall'input e mettilo nella cella 19 (**d**)
0000000000010000	**Carica** il contenuto della cella 16 (**a**) nel registro A
0001000000010001	**Carica** il contenuto della cella 17 (**b**) nel registro B
0110000000000000	**Somma** i registri A e B
0010000000010100	**Scarica** il contenuto di A nella cella 20 (**z**) (ris.parziale)
0000000000010010	**Carica** il contenito della cella 18 (**c**) nel registro A
0001000000010011	**Carica** il contenito della cella 19 (**d**) nel registro B
0110000000000000	**Somma** i registri A e B
0001000000010011	**Carica** il contenuto della cella 20 (**z**) (ris. parziale) in B
1000000000000000	**Moltiplica** i registri A e B
0010000000010100	**Scarica** il contenuto di A nella cella 20 (**z**) (ris. totale)
0101000000010100	**Scrivi** il contenuto della cella 20 (**z**) (ris. totale) in output
1101000000000000	**Halt**

I dati nei linguaggi di programmazione

Rappresentare l'informazione in Matlab

- Si usano principalmente le seguenti codifiche:
  - virgola mobile in doppia precisione (double, 8 byte)
  - caratteri (char, 1 byte)
  - 0/1 (logic, 1 byte)
- In Matlab non è necessario definire **esplicitamente** il tipo di codifica da utilizzare per ogni variabile.
- La codifica viene scelta dall'interprete sulla base del valore che viene assegnato ad una variabile:
  - double per i valori numerici
  - char per i caratteri
  - logic per i valori booleani
- Tuttavia è possibile anche specificare un particolare tipo di codifica per una variabile (ad esempio uint8 per i pixel delle immagini).

Rappresentare l'informazione in C

- A differenza di Matlab, in C è necessario definire in **maniera esplicita** il tipo (e quindi la codifica) di ogni variabile.
- I tipi fondamentali disponibili in C sono:
  - `int`, numeri interi con codifica CPL2
  - `float`, numeri in virgola mobile a singola precisione
  - `double`, numeri in virgola mobile a doppia precisione
  - `char`, caratteri con codifica ASCII