Codifica dell’informazione

La “macchina”da calcolo

- L’essenza dell’informatica sta nello scomporre l’informazione in “pezzi” elementari e la sua elaborazione in operazioni elementari
- Che cosa c`è di più elementare del **bit**, ossia di due possibili valori dell’informazione: {0,1}?
- Tutta l’informazione **discreta** può essere rappresentata come una sequenza di 0 e 1
  - quella continua può essere **approssimata**

Primi esempi di rappresentazione

- **Byte**: sequenza di 8 bit:
  (00000000, 00000001, 00000010, …, 11111111)
- Un byte può rappresentare i **numeri naturali** da 0 a 255 (= $2^8 - 1$):
  - zero = 00000000; 8 = 00001000; …; 255 = 11111111
- … e i numeri interi compresi fra –127 e 127, ossia fra $-(2^{(8-1)} - 1)$ e $(2^{(8-1)} - 1)$
  - primo bit = 0: numero positivo,
    primo bit = 1: numero negativo
  - attenzione: 0 = 00000000 e 0 = 10000000

Testo

- **Caratteri ASCII**
  (American Standard Code for Information Interchange)
  - sette bit usati per rappresentare 128 caratteri
    (ottavo per controllo)
- A ogni lettera (le maiuscole da A a Z, le minuscole da a a z), cifra (da 0 a 9) o separatore (usato per la punteggiatura o come operatore aritmetico) viene assegnato un numero naturale rappresentabile in forma binaria
  - ad esempio "A" viene codificata in ASCII come numero 65 e la sua forma binaria è 01000001; il separatore ";" viene codificato come 59 e la sua forma binaria è 00111011
- La stessa stringa di bit ha **diversi significati**, a seconda del tipo di informazione rappresentata!

Operazioni elementari

- **Inversione** di un bit: 0 → 1, 1 → 0
- **Somma** di due bit: 0+0 = 0, 1+0 = 1, 0+1 = 1, 1+1 = … 1
- **Prodotto** di due bit 0*0 = 0, 1*0 = 0, 0*1= 0, 1*1 = 1

- Se interpretiamo 0 come **Falso** e 1 come **Vero**, le operazioni di cui sopra possono essere viste come operatori logici
  - Inversione = Negazione, (NOT, ¬)
  - Somma = Somma logica, (OR, ∨)
  - Prodotto = Prodotto logico, (AND, ∧)

Aritmetica binaria

- Il fatto che l'informazione base sia il bit porta a codificare i numeri come sequenze di bit
- Ne consegue l'adozione della numerazione in base 2
  - 0 = 000; 1 = 001; 2 = 010; 3 = 011; ….
  - **algoritmo di conversione**:
    - 10/2 = 5; resto = 0
    - 5/2 = 2; resto = 1
    - 2/2 = 1; resto = 0
    - 1/2 = 0; resto = 1
  - 10_{10} : = 1010_2
  - $1010_2 = 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = 8 + 0 + 2 + 0 = 10_{10}$
- Da base 2 a base 8 (o 16) e viceversa è più facile, perché?

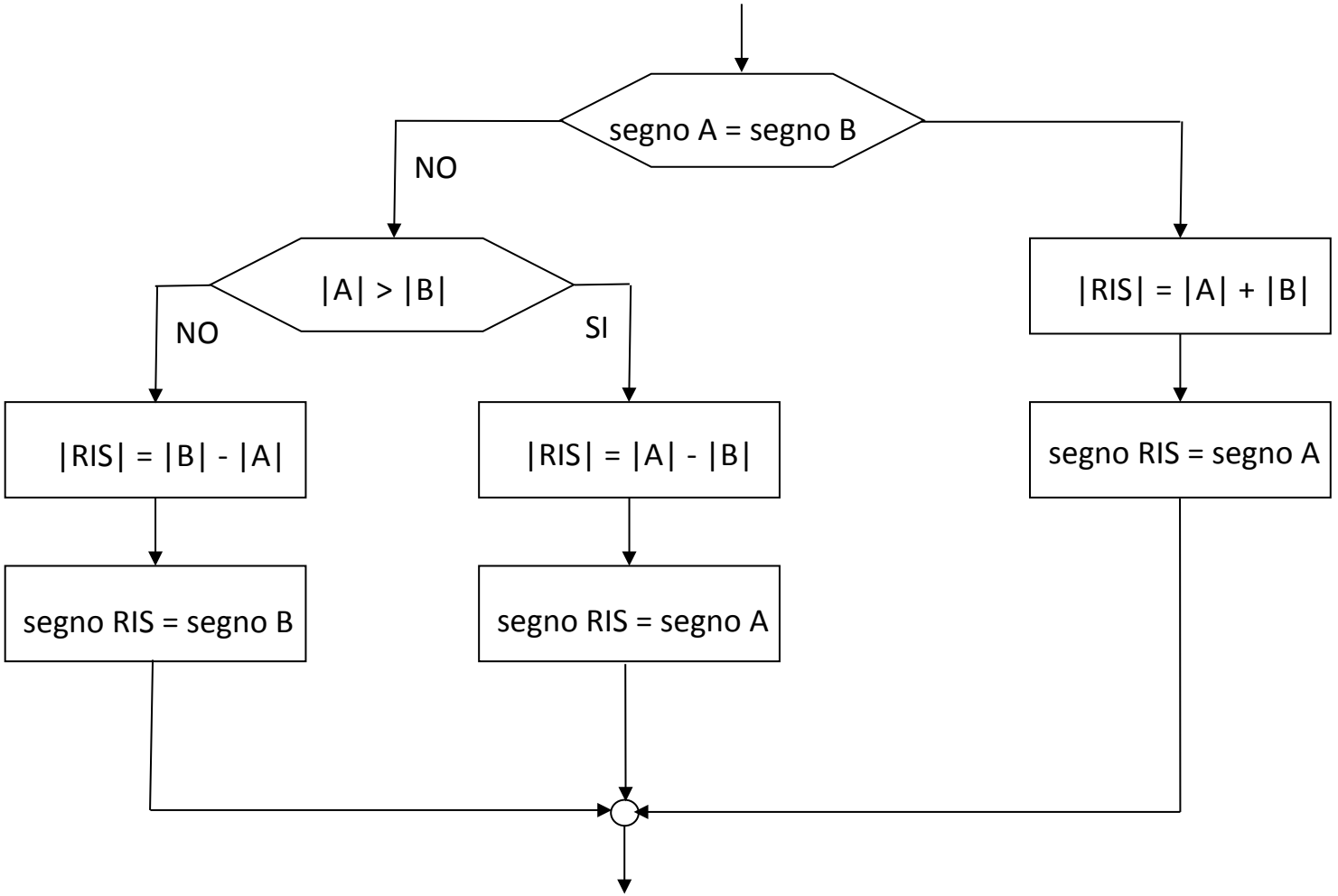
Somma cifra per cifra

riporto	11100	00110000000100
addendo	8731	10001000011011
addendo	5698	01011001000010
risultato	14429	11100001011101

Modulo e segno

- Dato un numero intero N, codificato su `n` bit
  - il bit più significativo rappresenta il **segno** (0 significa positivo e 1 negativo)
  - i restanti `n-1` bit rappresentano il **valore assoluto**
    - N = 6 → 3 bit + 1 per il segno → 0110
    - N = -6 → 1110
- Le operazioni aritmetiche elementari diventano già complicate
  - analisi del segno
  - confronto dei valori assoluti

Somma tra due numeri



Complemento a 1

- Codifica diversa per **semplificare** l'algoritmo di calcolo
- Non si distingue più il segno dal modulo
- Dato un numero N, il suo opposto si calcola complementando ad uno ad uno tutti i bit
  - N = 01001 → -N = 10110
- Somma e sottrazione richiedono solo sommatori e negatori (per il calcolo dell'opposto)
- Il risultato è corretto a meno di un 1 nel caso in cui si verifichi un riporto nella somma stessa
  - quindi si usa sempre una seconda somma per sommare il riporto generato (se necessario)

Esempio

- N = 011001 (+25) e M = 000011 (+3)
- N + M = 011001 + 000011 = 011100 (+28)

- K = 111100 (-3)
- N + K = 011001 + 111100 =
  (1)010101 → 010101 + 000001 = 010110 (+22)

- I due numeri devono essere rappresentati con lo stesso numero di cifre
- Sempre due somme
  - non è la soluzione ottima, ma è la meno costosa

Complemento a 2

- Ulteriore miglioramento,
  ma rappresentazione sempre più complicata
- Somme algebriche con una sola addizione
- Caratteristiche:
  - una sola codifica per il numero zero
  - numeri positivi → stessa codifica
  - numeri negativi
    - -N è quel numero che sommato a N produce una configurazione di tutti zero e un bit di riporto che si trascura
    - operativamente
      - complemento a 1 e poi si somma uno, oppure
      - si scorre il numero da destra a sinistra, lasciando inalterate le cifre fino al primo uno (compreso) e complementando le altre
- Notazione non simmetrica ($-2^{n-1} \leq N \leq 2^{n-1} - 1$)

Esempio

- N = 011001 (+25)
- -N = 100110 + 000001 = 100111 oppure
- -N = 100111
  - “salvo” solo il primo uno e complemento tutto il resto

- N = 011001 (+25) e M = 000011 (+3)
- K = -M = 111101 (-3)
- N + K = 011001 + 111101 = (1)010110 = 010110 (+22)

- M – N = 000011 + 100111 = 101010 (-22)
  - 010110 → +22

- Attenzione a leggere i numeri negativi!

Confronto

Codifica	Modulo e segno	Complemento a 1	Complemento a 2
0000	+0	+0	+0
0001	+1	+1	+1
0010	+2	+2	+2
0011	+3	+3	+3
0100	+4	+4	+4
0101	+5	+5	+5
0110	+6	+6	+6
0111	+7	+7	+7
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

Numeri razionali

- **Numeri razionali** contenenti una parte intera e una frazionaria che approssimano il numero reale con **precisione arbitraria**
  - notazione **in virgola fissa**: si codificano separatamente la parte intera e la parte frazionaria: ad esempio: 8.345:
    - primo byte (rappresentazione dell'intero 8) = 00001000
    - secondo byte (rappresentazione della parte frazionaria 0.345) = 01011000
    - però: 0*2^-1+1*2^-2+0*2^-3+1*2^-4+1*2^-5=0,34375 → approssimazione/troncamento

0.587_{10}

- 0.587 × 2 = 1.174: parte frazionaria 0.174 e parte intera 1
- 0.174 × 2 = 0.348: parte frazionaria 0.348 e parte intera 0
- 0.348 × 2 = 0.696: parte frazionaria 0.696 e parte intera 0
- 0.696 × 2 = 1.392: parte frazionaria 0.392 e parte intera 1
- 0.392 × 2 = 0.784: parte frazionaria 0.784 e parte intera 0
- 0.784 × 2 = 1.568: parte frazionaria 0.568 e parte intera 1
- 0.568 × 2 = …
- = 0.1001 (con quattro cifre binarie dopo la virgola) o
- = 0.100101 (con sei cifre binarie dopo la virgola)

Numeri reali

- Approssimati da razionali
- **Virgola mobile**:
  - **mantissa** (o significante) e **esponente** (o caratteristica): $r = m \times b^n$
  - la quantità di cifre nella mantissa determina la **precisione**
- Esempio
  - con b = 10: -331.6875 viene rappresentato con m = -0.3316875 e n = 3

Numero normalizzato

- Un numero in virgola mobile si dice **normalizzato** se la virgola della mantissa è posizionata subito a sinistra della prima cifra diversa da 0
  - $+0.45676 \times 10^2$ normalizzato
  - $+0.0456 \times 10^4$ non normalizzato

Virgola mobile in binario

- Mantissa ed esponente sono codificati in bit
  - più un eventuale bit di segno per la mantissa
- Esempio
  - con b = 2, bit di segno della mantissa 0, mantissa 1011 e caratteristica 01010 viene interpretato in base decimale come: $0.6875 \times 2^{10} = 0.6875 \times 1024 = 704.01$

Lo standard IEEE

- 4 formati per la rappresentazione dei numeri reali che differiscono per il numero totale di bit utilizzati: i più diffusi:
  - a precisione singola, con 32 bit
  - a doppia precisione, con 64 bit