# Rappresentazione dei dati in memoria

# La memoria

- Una memoria deve essere un insieme di oggetti a più stati. Questi oggetti devono essere tali che:
  - le dimensioni siano limitate
  - il tempo necessario per registrare o leggere un'informazione sia minore possibile
  - l'energia necessaria sia minima
  - la registrazione e la lettura siano affidabili

# Elementi di memoria

- Le ultime 4 condizioni non sono realizzabili con dispositivi meccanici, ma solo con sistemi magnetici e/o elettronici.
- Non esistono però sistemi magnetici o elettronici con più di due stati (mentre questo è possibile con dispositivi meccanici).
- Per gli elementi di memoria dei calcolatori vengono quindi utilizzati elementi non meccanici a due stati.

# Numeri e loro codifica

Per non confondervi ricordate sempre che:

Un numero ha un valore quantitativo intrinseco, indipendentemente dalla rappresentazione usata per esprimerlo in cifre.

# Codifica e Basi

- Noi usiamo abitualmente la base 10 per esprimere in cifre i numeri, forse perché abbiamo 10 dita. Per rappresentare il numero 12 ad esempio contiamo fino a 10, esauriamo le dita, scriviamo quindi un 1 "nella colonna delle decine" e ricominciamo a contare. Contiamo 2 e lo scriviamo "nella colonna delle unità", ottenendo appunto 12.
- Supponiamo di avere solo un dito, come una memoria del calcolatore: come rappresenteremmo il numero 5?

# Rappresentazione in base 2

- Cominciamo a contare: 0, 1, 2, ma a 1 abbiamo già finito le dita, scriviamo quindi 10 (che vale 2) e continuiamo a contare.
- 3 sarà espresso come 11
- 4 sarà espresso come 100
- 5 sarà dato da 101
- Formalmente

$$(5)_{10} = 5*10^0 = 1*2^2 + 0*2^1 + 1*2^0 = (101)_2$$

# Sistema di numerazione posizionale

- La rappresentazione delle informazioni numeriche con il sistema posizionale è definita dall'espressione:

$$(\ldots a_2\ a_1\ a_0\ .\ a_{-1}\ a_{-2}\ \ldots)_b =$$

$$\ldots + a_2\ b^2 + a_1\ b^1 + a_0\ b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} + \ldots$$

- in cui b è la base, le cifre a sono elementi di un insieme costituito da b simboli diversi, il punto separa gli elementi associati a termini con esponente negativo da quelli associati a termini con esponente positivo o nullo.

# Basi di uso più comune

- Base 10 (rappresentazione decimale)
- Base 2 (rappresentazione binaria)
- Base 8 (rappresentazione ottale)
- Base 16 (rappresentazione esadecimale)
  - NB mentre per rappresentare un numero nelle prime 3 basi sono sufficienti i 10 caratteri 0-9, in base esadecimale servono degli altri simboli per rappresentare i numeri 10,11,12,13,14 e 15, questi sono A B C D E e F

# Conversioni tra basi: esempi (1)

- Il numero binario 10011 convertito in base 10 vale 1*1+1*2+0*4+0*8+1*16=19
- Il numero ottale 5071 convertito in base 10 vale 1*1+7*8+0*64+5*512=1+56+2560=2617
- Il numero esadecimale 48F convertito in base 10 vale 15*1+8*16+4*256=15+128+1024=1167

# Conversioni tra basi: esempi (2)

- Come convertiamo il numero decimale 19 in base 2? Esprimendolo come una combinazione lineare di potenze di 2

  $$1*1 + 1*2 + 0*4 + 0*8 + 1*16$$

  ovvero eseguendo divisioni successive per 2, fermandosi al risultato intero e registrando il resto ...

| Dividendo | Risultato | Resto |
|---|---|---|
| 19 | 9 | 1 |
| 9 | 4 | 1 |
| 4 | 2 | 0 |
| 2 | 1 | 0 |
| 1 | 0 | 1 |

- E ricordando che il primo resto è il coefficiente del termine di ordine più basso del polinomio avremo 10011.

# Conversioni tra basi:esempi (3)

- Conversione in base 8 del numero decimale 2617: questa volta si procede mediante divisioni successive per 8

| Dividendo | Risultato | Resto |
|---|---|---|
| 2617 | 327 | 1 |
| 327 | 40 | 7 |
| 40 | 5 | 0 |
| 5 | 0 | 5 |

E quindi la rappresentazione ottale cercata è 5071

# Conversioni tra basi: esempi (4)

- Conversione del numero decimale non intero 15.375 in base 2:
  - Parte intera: 15

  ovvero 1111 in base 2
  - Parte frazionaria:0.375 = 0*1/2 + 1*1/4 + 1*1/8

  ovvero 0.011 in base 2

  Quindi 15.375 in base 10 corrisponde a 1111.011 in base 2

# Conversioni tra basi: esempi (5)

- Metodo generale per convertire la parte frazionaria: moltiplicazioni successive per la nuova base registrando le parti intere. In questo caso N=0.375 moltiplicatore=2
- Moltiplicando risultato parte intera

  0.375 0.750 0

  0.750 1.500 1

  0.500 1.000 1

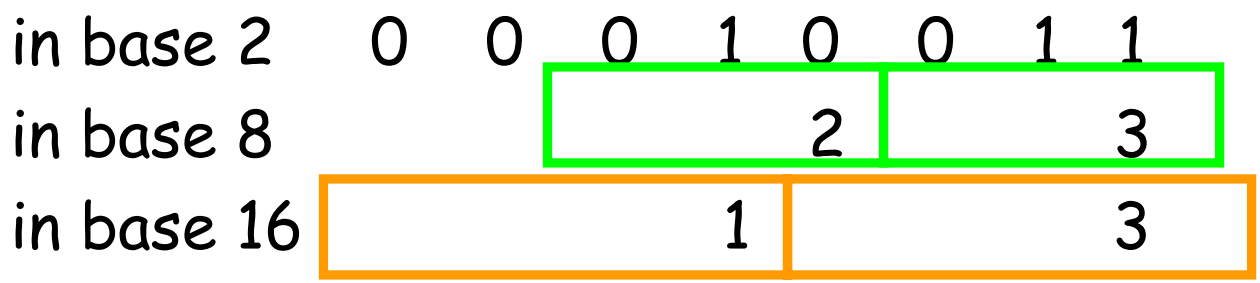
  0

  E ritroviamo 0.011

# Conversioni tra basi: esempi (6)

- Caso particolarmente semplice: le conversioni tra basi che sono potenze di 2 (binaria, ottale, esadecimale). Infatti 3 bit (valori compresi tra 0 e 7) corrispondono ad una cifra ottale e 4 bit (valori compresi tra 0 e 15) corrispondono ad una cifra esadecimale.

Il numero decimale 19 si scrive



# Operazioni fra variabili numeriche binarie

- Addizione:
  - 0+0=0, 0+1=1, 1+0=1, 1+1=10
- Sottrazione:
  - 0-0=0, 1-0=1, 1-1=0, 10-1=1
- Moltiplicazione:
  - 0*0=0, 1*0=0, 1*1=1
- Divisione:
  - 0/1=0, 1/1=1, 10/1=10

# Esempi di operazioni fra variabili numeriche binarie

- 11010101+1100011=100111000

(213+99=312 in base 10)

- 1010011-101001=101010

(83-41=42 in base 10)

- 10110*11=1000010

(22*3=66 in base 10)

- 1001010/101=1110 con resto di 4

(74/5=14.8 in base 10)

NB 4/5=0.8 ha un numero infinito di cifre in rappresentazione binaria.

# Rappresentazione in memoria dei numeri

- Disponiamo di due soli stati e quindi non possiamo inserire un separatore tra due numeri in memoria (servirebbe un terzo stato).
- I numeri in memoria quindi occupano una quantità fissata di bit consecutivi (voce di memoria) qualunque sia il loro numero di cifre significative.
- Il numero di bit di una voce varia a seconda dell'elaboratore ma è sempre un multiplo di 4 (generalmente di 8).

# Rappresentazione in memoria dei numeri interi

- Per registrare un numero intero in una voce dobbiamo registrarne sia il segno che il valore.
- Per il segno ci sono due possibilità: riservagli un bit, per esempio il primo da sinistra, oppure utilizzare il suo complemento . Per i numeri interi si utilizza generalmente questa seconda soluzione.
- La complementazione consiste nello scambiare tutti gli 0 di un numero con degli 1 e viceversa e nell'aggiungere un 1 al bit meno significativo (quello più a destra).

# Rappresentazione in memoria dei numeri interi

- Suppondendo di lavorare con una voce a 32 bit (4 byte) il numero 19 sarà rappresentato da

00000000 00000000 00000000 00010011

- Mentre –19 sarà

11111111 11111111 11111111 11101101

- Con questa rappresentazione dei numeri negativi è immediato avere (19)+(-19)=0 !

  00000000 00000000 00000000 00010011 +

  11111111 11111111 11111111 11101101 =

0 00000000 00000000 00000000 00000000

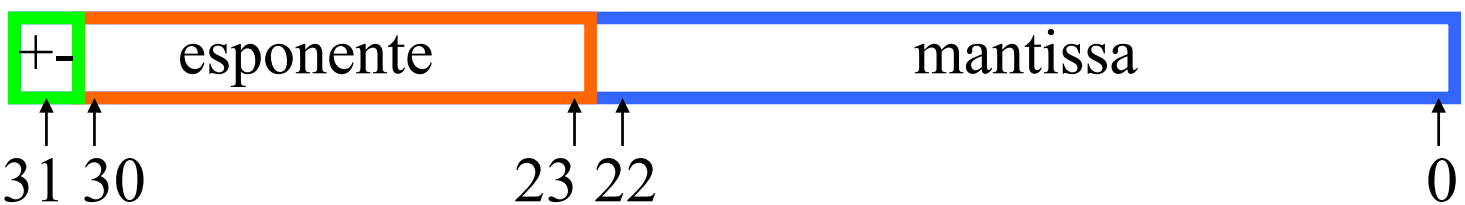
# Rappresentazione in memoria dei numeri reali

- Per registrare un numero reale binario, per esempio +110.010011 sorge il problema di non poter indicare la posizione del punto decimale che è variabile (*floating point*).
- La soluzione generalmente adottata consiste nell'esprimere il numero in forma esponenziale "normalizzata"e registrare separatamente nella voce l'esponente e la mantissa.

$$+110.010011 = +1.10010011 * 2^{2}$$

# Rappresentazione in memoria dei numeri reali

- Lo standard IEEE 754 prevede nel caso di voci a 32 bit
  - che il bit più significativo (bit 31) indichi il segno della mantissa (0=+, 1=-)
  - Che i successivi 8 bit (bit 30-23) contengano la rappresentazione dell'esponente nella forma "eccesso 127" (ovvero esponente+127)
  - I rimanenti 23 bit (bit 22-0) contengano la parte frazionaria della mantissa



# Rappresentazione in memoria dei numeri reali

- Poichè la mantissa è "normalizzata" è sempre del tipo 1.xxxxxx e quindi non è necessario rappresentare l'1 prima del punto. La mantissa è dunque espressa con 24 bit ma solo i 23 meno significativi sono rappresentati.
- Fa eccezione il numero 0 per il quale tutti e 32 i bit sono nulli.
- La rappresentazione dell'esponente può assumere valori compresi tra –127 e 128, estremi però riservati a indicare condizioni eccezionali.

# Rappresentazione in memoria dei numeri reali: esempio

- Rappresentiamo il numero decimale 0.03125
  - la rappresentazione in forma esponenziale normalizzata è

    $1*2^{-5}$

    -il segno è +, quindi il bit più significativo sarà 0

    - l'esponente sarà registrato come 127-5=122 che in forma binaria si scrive 01111010

    - la parte frazionaria della mantissa normalizzata è 0

-Otteniamo quindi

00111101 00000000 00000000 00000000

in notazione esadecimale (più compatta) 3D000000

# Numeri rappresentabili (32 bit)

- Numeri interi compresi tra

$-2^{31} = 10000000\ \ 00000000\ \ 00000000\ \ 00000000$

$2^{31} - 1 = 01111111\ \ 11111111\ \ 11111111\ \ 11111111$

Ovvero tra i numeri decimali -2147483648 e 2147483647

- Lo zero e alcuni numeri reali compresi tra

$-2^{-126}$ $\rightarrow (2 - 2^{-23}) * 2^{127}$

- Infatti la minima distanza tra due numeri reali consecutivi è data dal bit meno significativo della mantissa per 2 elevato all'esponente.
- Se il modulo di un numero è maggiore del massimo numero rappresentabile si ha un *overflow*, se è compreso tra zero ed il più piccolo numero positivo rappresentabile si ha un *underflow*.

# Variabili logiche

- La logica si occupa di proposizioni o enunciati che possono assumere due soli stati, vero o falso, che sono quindi rappresentabili in un sistema binario.
- Scegliamo la convenzione che assegna i valori 1 ad una proposizione vera e 0 ad una falsa.
- Le operazioni logiche fra tali proposizioni dovranno riprodurre le funzioni (operatori logici) "e", "o", "non" e le loro combinazioni.
- Siano A e B due variabili logiche definiamo alcune operazioni tra di esse.

# Operazioni logiche: AND

- Prodotto logico: corrisponde all'"e" del linguaggio comune ed è indicato col termine AND. Si indica con P=A.B (ma si codifica in modo diverso a seconda del linguaggio di programmazione!). P è vero solo se lo sono sia A che B. La "tabella della verità" è:

| A | B | P=A.B |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

# Operazioni logiche: OR

- Somma logica inclusiva: corrisponde all'"o" del linguaggio comune ed è indicata col termine OR. Si indica con S=A+B (ma si codifica in modo diverso a seconda del linguaggio di programmazione!). S è vera solo se lo è almeno una delle variabili A e B.

| A | B | S=A+B |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

# Operazioni logiche: XOR

- Somma logica esclusiva: corrisponde all'"o" restrittivo del linguaggio comune ed è indicata col termine XOR. Si indica con S=A/B (ma si codifica in modo diverso a seconda del linguaggio di programmazione!). S è vera solo se lo è una sola delle variabili A e B.

| A | B | S=A+B |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |
| 1 | 1 | 0 |

# Operazioni logiche: NOT

- Inversione logica: corrisponde al "non" del linguaggio comune ed è indicata col termine NOT. Si indica con $I=\overline{A}$, o I=-A (ma si codifica in modo diverso a seconda del linguaggio di programmazione!). I è vera solo se A è falsa

| A | $I=\overline{A}$ |
|---|---|
| 1 | 0 |
| 0 | 1 |