Binario

Thursday, 17 August 2023 09:17

- I computer possono contare solamente con 1 o 0, questo per via dell'elettricità: o c'è
  - Per trasformare da binario in decimale, sobbiamo la moltiplicazione della cifra p
    Es:
    $1^a0^b1^c1_2 = (1^a * 2^3) + (0^b + 2^2) + (1^c + 2^1) + (1 * 2^0) = 8 + 0 + 2 + 1 =$
    (Le lettere sopra ai numeri servono solamente per farci capire cosa stiamo i
    A seconda del numero di bit che il computer possiede, in memoria verrà visu
    diverso, es:
    Con un computer di 8 bit, 1011 verrà mostrato come 0000 1011
    Con un computer da 16 bit, 0000 0000 0000 1011
    - Quando noi usiamo tutti i bit per mostrare il numero, noi diciamo che è
    - Quando usiamo il bit più a sinistra per specificare il segno, invece è sig
      Es:
      0111 1111 1111 1111 1111 1111 1111 1111 = 2 147 483 647
      1000 0000 0000 0000 0000 0000 0000 0000 =-2 147 483 647
      Perchè questo?
      Trasformiamo da binario in decimale manualmente. Abbiamo detto ch
      segno, quindi
      $(1 * -2^{31}) + (0 * 2^{30}) + (0 * 2^{29}) \ldots = -2^{31} = -2\ 147\ 384\ 647$
    - Per negare un numero basta sommare 1 e invertire tutti i valori
      Es.
      $2_2 = 0000\ 0010$
      Invertiamo: 1111 1101 +
      Ed ora sommiamo 1
      =======
      $-2_2 =$ 1111 1110
    - Per convertire da 16 bit a 32 bit basta aggiungere 16 zeri a sinistra
    - Per fare da negativo a positivo, sottraiamo 1 e facciamo l'inverso
      1111 1110 -
      1
      ======
      1111 1101
      ======
      0000 0010 = 2
    - Questa conversione viene chiamata complemento a 2
      - Senza sommarre/sottrarre 1 diventa complemento a 1
  - Il binario può essere trasformato in decimale così:

Eca8
$e_{16} = 14_{10} = 1110$
$c_{16} = 12_{10} = 1100$
$a_{16} = 10_{10} = 1010$
$8_{16} = 8_{10} = 0100$
E' facile ora fare l'inverso

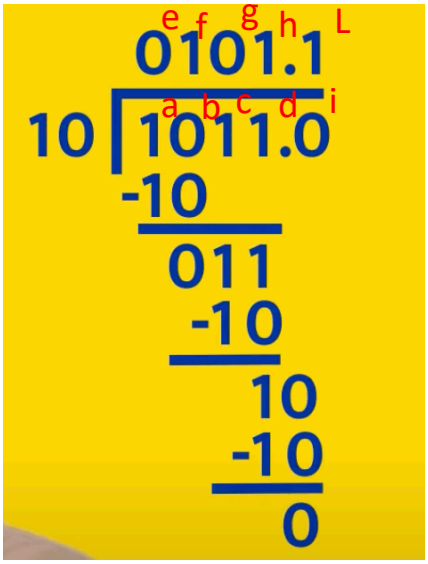
- Operazioni logiche tra i binari:

Operazione	C	MIPS	Spiegazione
Shift left	<<	Sll	Sposta i bit a sinistra, moltiplica per 2^x
Shift right	>>	Srl	Sposta i bit a destra, divide per 2^x
Bit by bit AND	&	And, andi	
Bit by bit OR	|	Or, ori	
Bit by bit not	~	nor	

- Addizione/Sottrazione
0000 0111 +
0000 0110
======
0000 1101 (E' come alle elementari)

0000 0111 +
1111 1010
=======
0000 0001 (overflow, 7+(-6)=1)
Nota: se usassimo add ci verrebbe a creare un eccezione di overflow
Per ovviare ussare addu (add unsigned)

- Divisioni



Ragionamento:
1) 10 è minore di 1^a quindi aggiungiamo uno 0^e sopra

1) 10 è minore di 1^a quindi aggiungiamo uno 0^b sopra
2) 10 è uguale di 1^a0^b quindi aggiungiamo aggiungiamo un 1^f sopra e so
3) Portiamo giù 1^c, 01 è minore di 10 quindi portiamo sotto 1^d ed aggiun
4) 011 è più grande di 10 quindi aggiungiamo 1 sopra e sottraiamo per 1
5) 1 è più piccolo di 10, però non possiamo portare nulla sotto siccome ab
   Allora mettiamo la virgola ed aggiungiamo uno 0
6) Portiamo giù 0^i e notiamo che 10 è uguale a 10, quindi portiamo su 1^L

Risultato: $1011 = 101.1*10+0$

- Floating point

  Formato da: segno, mantissa, esponente
  IEE 754 usa questa quantità:

Sign bit	Esponente	Mantissa
1 bit	8 bit	23 bits

  Procedura con esempio:
  19.59375
  1) Determinare il sign bit, quindi positivo/negativo
     E' positivo quindi 0
  2) Convertire il numero il binario
     - La parte intera si trasforma normalemte
       $19 = 10011$
     - La parte con la virgola si trasforma moltiplicando
       $.59375 * 2 = 1.1875$ | 1
       $.01875 * 2 = 0.375$ | 0
       $.0375 * 2 = 0.75$ | 0
       $0.75 * 2 = 1.5$ | 1
       $0.5 * 2 = 1$ | 1

     Quindi 10011,10011
  3) Normalizzare la mantissa
     Dobbiamo portare la virgola a sinistra fino ad arrivare all'1
     $1{,}0011\ 10011 * 2^4$
  4) Calcolare il bias exponent (aggiungere 127) e poi trasformare in binari
     $4 + 127 = 131 = 10000011$
  5) Rimuovere 1 dalla mantissa
     $1{,}0011\ 10011 = 0011\ 10011$
     Aggiungere abbastanza 0 fino ad arrivare a 24 bit

Sign bit	Exponent	Mantissa
0	10000011	00111001100000....0000

  Nota: noi sopra abbiamo usato single precision
  Double precision ha i seguenti bits:

Sign bit	Esponente	Mantissa

Sign bit	Esponente	Mantissa
1 bit	11 bit	20 bits

Per la somma in floating point cerchi di portare l'esponente più piccolo semp
più grande