Codifica binaria

Nei calcolatori i numeri sono rappresentati a variabile fissa (2^k) con k Bit.

0 1 0 1 1 0 0 1 $\downarrow$ CODIFICA

$\ldots 2^6\, 2^4\, 2^3\, 2^2\, 2^1\, 2^0$

$\underset{\text{VALORE}}{V} = 1 \cdot 2^0 + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^6 = 89$

Per rappresentare i caratteri si usano differenti codifiche, ad esempio si adotta la codifica ASCII, in cui ad ogni carattere corrisponde un numero (o una certa sequenza numerica).

es.

A $\rightarrow$ 1
B 2
C 3
D 4
⋮ ⋮

Per rappresentare i numeri *relativi* (= con segno) posso riservare uno spazio (bit) al **segno**, che va a moltiplicare il **valore assoluto** (o modulo). Il numero relativo è dunque rappresentabile secondo la formulazione Modulo e Segno:

S

$\underset{\text{SEGNO}}{\boxed{\ }}\ \underbrace{0\,1\,0\,1\,1\,0\,0\,1}_{\text{MODULO}}$

Formulazione:

$$(-1)^{\overset{\text{SEGNO}}{S}} \cdot \underset{\text{VALORE}}{V}$$

Dove 'S' indica PARI o DISPARI (ovvero il segno del numero)

Rappresentazione C-1 e C-2

domenica 27 settembre 2020 16:44

Per venire incontro ai problemi della precedente formulazione, si utilizza un nuovo genere di rappresentazione: "complemento a 1" e "complemento a 2".

Complemento a 1 (c-1): inversione delle cifre binarie

Complemento a 2 (c-2): aggiunta al c-1 di un numero 1

01011001 = 89

10100110 + → rappresentazione C-1 di -89

00000001 = → trasformo in C-2 (aggiungo 1)

10100111 → rappresentazione in C-2

Applicazioni di C-2

domenica 27 settembre 2020 16:45

La rappresentazione c-2 è semplice ed efficiente per rappresentare l'algoritmo di somma dei numeri senza segno

RIPORTO (1+1 IN BINARIO = 0 CON RIPORTO)

11 +	1011	ALGORITMO DI SOMMA
2 =	0010	
13	1101	
	8+4+1 = 13	

Se ora volessi aggiungere il segno (usando la rappresentazione modulo e segno) necessito di un bit in più. Mi accorgo presto che l'operazione fra due numeri opposti di segno non è più una somma ma una differenza.

S

0 1011 +

1 0010 =

1 1101

L'algoritmo di somma precedente NON BASTA

Se utilizziamo la rappresentazione c-2 possiamo non considerare il segno:

11 → 0 1011

-2 → 0 0010 → considero il valore in modulo (associato) → 2

C-1: 1 1101 + 0 0001 =

C-2: 1 1110

Procedo col calcolo di somma:

+11 +	0 1011 +
−2 =	1 1110 =
+9	~~1~~ 0 1001 = +9 → 8+1 = 9

NON CI STA → troppi pochi BIT

Mi basta quindi rappresentare in C-2 solo il numero negativo

Mi basta quindi rappresentare in C-2 solo il numero negativo.
Viceversa, per tornare al numero positivo procedo nel seguente modo:

$(-2 \rightarrow +2)$

-2 (in C-2)

$$\begin{array}{r} 11110 \\ 00001 + \\ 00001 = \\ \hline 00010 \end{array}$$

inverto (C-1)

+1

+2 (in C-2)

Più velocemente posso scansionare i numeri da destra: per ogni zero scrivo zero, per l'uno scrivo uno.
Appena trovo il primo uno, inverto le cifre successive (andando verso sinistra):

da -2: 11110 scansiono e ottengo:

$\underbrace{00}_{\text{inverto}}0\underbrace{1}_{1° \text{ uno}}\underbrace{0}_{\text{zero}} = 2$

alternativa x cambiare segno

Concetto di overflow

domenica 27 settembre 2020 17:07

Durante il calcolo di somma può capitare che la rappresentazione di un numero sia errata, in quanto il numero eccede lo spazio predestinato al risultato:

$$
\begin{array}{lr}
 & S \\
+11 & 0\,1\,0\,1\,1\ + \\
+12 & 0\,1\,1\,0\,0\ = \\
\hline
 & 1\,0\,1\,1\,1 \\
 & \neq 23
\end{array}
$$

L'ultima posizione binaria riservata al segno assorbe la cifra più significativa del risultato. Vi è una perdita di informazioni, con conseguente risultato sbagliato.

RISULTATO SBAGLIATO !

(viene fuori un numero < 0 !)

Overflow

il risultato da rappresentare è troppo grande rispetto alla rappresentazione FISSA che ho scelto di utilizzare

l'OVERFLOW NON si verifica con la somma di due numeri opposti (= differenza)

Rappresentazione a eccesso 2^m

domenica 27 settembre 2020 17:19

Un altro metodo di rappresentazione è quella ad eccesso 2^m con m = numero di cifre binarie considerate (n_bit) -1 (spazio riservato al segno).

es 5 BIT

| | | | | |

S MANTISSA

quindi $2^{5-1} = 2^4 = 16$

SOTTRO TALE COSTANTE ai valori da calcolare così da trasformare il valore ottenuto.

$V + 16 = V'$

rappresento V' in 5 BIT

es. 5 BIT

- +11 $\xrightarrow[\text{CODIFICO}]{}$ 11 + 16 = 27 e rappresento in binario, come VALORE ASSOLUTO. (NO SEGNO)

S

| 1 | 1 | 0 | 1 | 1 | = 27

eccesso in MANTISSA

Se lo penso con segno e modulo

| 0 | 1 | 0 | 1 | 1 |

Rappresentazione uguale a meno del BIT DEL SEGNO

Rappresentazione In C-2

- −2 → −2 + 16 = 14

0 1 1 1 0 = 14 → SOVRASCRIVO IL BIT DEL SEGNO CON QUELLO DI −2 (quindi 1)

$\boxed{1}$ 1 1 1 0 = −2

resto invariato

N.B. Equivale alla rappresentazione C-2:

0 0 0 1 0 → 1 1 1 1 0 (met. veloce)

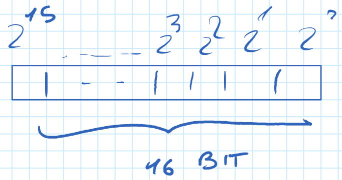
+2 −2

Rappresentazione in virgola fissa

domenica 27 settembre 2020 17:36

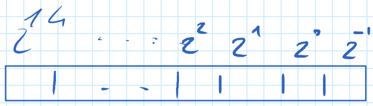
Per rappresentare i numeri razionali si usa la rappresentazione in virgola fissa (fixed point), secondo la formulazione:

$$0 < \text{valore costante} < 1$$



Posso così rappresentare $1/2$, $1/4$, $1/8$, $1/16$ ecc...

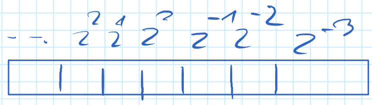
cambiando il metodo di codifica:



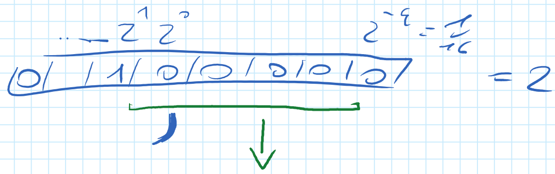
→ rinuncio a rappresentare i valori numericamente più grandi

$\cancel{2^{15}} \rightarrow 2^{14}$

Posso via via "spostare la virgola di separazione della parte frazionaria" verso sinistra, riservando più posti per la parte decimale, rendendo l'eventuale risultato meno approssimato di quello che è già.

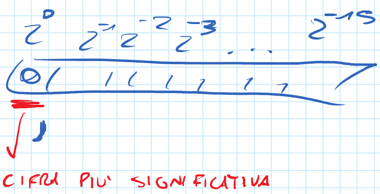


Bisogna però fissare la virgola. Essa non è un concetto proprio del computer. Se devo rappresentare con un dettaglio sino a 1/16, posiziono la virgola sul 5° bit.



Quello che era un 32, fissando ora la virgola, diventa un 2. Ricordiamo che per il computer rimane un 32, ma secondo una codifica a me nota, posso leggerlo come 2.

Se volessi rappresentare una probabilità, cerco di rappresentare i valori frazionari fra 0 e 1, ponendo una virgola sull'ultimo bit (quello più a sinistra)



Alcuni numeri frazionari sono rappresentabili esattamente, mentre altri (es. 1/3) non esattamente, si applica dunque un'approssimazione (per eccesso o per difetto).
Ricordiamo inoltre che è possibile sommare numeri interi e numeri frazionari, applicando la codifica a virgola fissa anche al numero intero (e dopo la virgola avrò N zeri): es avendo 32 bit, posso riservarne 16 alla parte intera e 16 alla parte decimale.

$$10{,}0000\ldots +$$
$$\underbrace{00}_{\text{parte intera}}{,}\underbrace{0101}_{\text{parte decimale}}\ldots =$$

Rappresentazione in virgola mobile

domenica 27 settembre 2020 17:58

Un altro metodo di rappresentazione dei numeri con parte decimale è la rappresentazione in virgola mobile (floating point), secondo la formulazione:

$V = m \cdot 2^{e}$ → valore che cambia

- m = mantissa
- e = esponente

RISERVO: $\underbrace{\quad}_{d}$ $\underbrace{\quad}_{m}$

d BITS per 'e' — m BITS per la mantissa

es. con 16 BIT: 4B × e, 12B × m

e				m					
0	1	0	0	1	0	1	0	0	…

$= \left(1 + \frac{5}{8}\right) \cdot 2^{4}$ → $-4+8=4$

se riservo 4 BIT × e, allora ho una codifica eccesso a 8 (posso scrivere valori da -8 a 7)

→ Includo così la notazione a virgola fissa, con cui pongo la virgola al 12° bit

es. 32 BIT

e: 8 BIT — m: 24 BIT

calcolo l'esponente con la rapp. eccesso 2^{n} → $2^{8\,BIT-1} = 128$ (costante da sommare a V) → V'

Riservando uno spazio sia all'esponente che alla parte decimale posso rappresentare le mie cifre decimali indipendentemente dalla parte non decimale, la cui grandezza viene definita dall'esponente. Quindi ho la possibilità di rappresentare la mantissa su una precisione definita, precisione che posso riservare anche per i numeri grandi.

→ Posso rappresentare numeri grandi e piccoli, cosa che la rappresentazione in fixed point non mi permetteva

Canonicità

domenica 27 settembre 2020 18:12

Per rappresentare il numero 1, posso usare m = 1 ed e = 0

$\rightarrow v = m \cdot 2^e = 1 \cdot 1 = 1$

ecc. 8

4B 12B

1 | 0 | 00 | 1, 0 0 0 ...

16 B

0+8 = 8

Ma posso anche dire m = 1/2 ed e = 1

$\rightarrow v = \frac{1}{2} \cdot 2^1 = 1$

1001 | 0 1 0 00 ...

2^{-1}

1+8 = 9

ESCLUDO

La rappresentazione del numero 1 dunque non è canonica, in quanto vi sono più modi per rappresentare il numero 1, quindi senza applicare la decodifica (quindi applicando la formula v = m * 2^e), non posso dire con certezza se le due rappresentazioni sono uguali o meno, come appena mostrato.
Posso dunque inserire la **condizione di normalizzazione** con cui elimino tutte le altre possibilità di rappresentazione del numero al di fuori di quella scelta

Posso quindi affermare che la prima cifra binaria di m DEVE assumere valore 1, quindi solo la rappresentazione con m = 1 ed e = 0 è la rappresentazione del valore 1, le altre possibilità non sono normalizzate.