Architettura degli Elaboratori - Brutta

Andrea Malvezzi

09 Ottobre, 2024

Contents

Differenze tra nostri calcolatori e ALU HACK:
La ALU dell'Architettura HACK è a 16 bit.
No bit di overflow, occorre andare a guardare i bit segno e fixare l'errore a mano.
Esercizio per capire meglio: implementare circuito di mul con ALU HACK

1 Struttura HACK

1.1 Adder

Circuito per fare le somme, prendendo due input (entrambi a 16 bit) e produce un out (a 16 bit) e un bit di riporto.
Noi sappiamo implementarlo, la somma effettiva sono xor, il riporto è una and (si ha riporto solo se son entrambi 1).
Vedi appunti cartacei per implementazione di Half-adder e poi full-adder a 3 bit con 2 Half-adder. Ora abbiamo quasi tutti gli strumenti per costruire l'ALU HACK.
Nella tabella dei bit di controllo, zx/nx e zy/ny fanno la stessa cosa, ovvero: se z(x/y) (inteso come una AND tra le due variabili), allora poni x=0. Se n(x/y), allora poni x = !x. Se f è vero, fa la OR, altrimenti la AND. Se ng è vero, nega l'output.
Vedi schema uso dei control bit nell'architettura Hack da appunti cartacei.
La OR da questo circuito si può fare negando i due ingressi e facendo la AND (De Morgan): $!(\overline{x} \cdot \overline{y}) = x+y$. Per la sottrazione vedi esempio cartaceo. FARE SECONDO PROGETTO e riassumere fino a slide 16 la rappresentazione dell'informazione.

2 Slide 16 rappresentazione informazione

Numeri troppo grandi -¿ overflow; troppo piccoli -¿ underflow.
Spesso quindi non potrò rappresentarli, così come anche talvolta numeri tra altri due numeri (tra il 10 e l'11 magari manco 10,00001 etc...).

2.1 Floating point

La codifica Floating point si basa sul rappresentare un numero (n) tramite altri due numeri (f -¿ frazione o "mantissa" ed e -¿ esponente o "caratteris-

tica").
Di norma si normalizza la frazione, ovvero la prima cifra del decimale non può essere 0. Quindi ad esempio, avendo base 2 e in notazione di eccesso 64:

$$0\,1010100, 0000000000011011 = 2^{20} \cdot \dots = 432$$

Ho 11 zeri dopo la virgola, quindi sposto di 11 cifre a sinistra e rimuovo 11 dall'esponente (parte prima della virgola).
A proposito di numero prima della virgola, traducendolo otterrei 84, ma essendo in eccesso 64 devo togliere 64 dal risultato, ottenendo 20 (che è l'esponente). Quindi ora so che il nuovo esponente sarà 20 - 11 = 9, che nella codifica dovrò ricordare di incrementare per l'eccesso, quindi 9 + 64 = 73. Inoltre, dopo la virgola partirò dal primo 1 presente nella codifica originale e scriverò zeri dopo, per riempire eventuali spazi (devo comunque avere 16 bit).
Quindi la nuova codifica sarà:

$$0\,1001001, 1101100000000000 = 2^{9} \cdot (1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5}) = 432$$

Un altro esempio, sta volta in base 16 (sempre con eccesso 64):

$$0\,1000101, 0000\ 0000\ 0001\ 1011 = 16^{5} \cdot (1 \cdot 16^{-3} + B \cdot 16^{-4}) = 432$$

Qui i bit son raggruppati in cifre esadecimali, quindi occorrerà traslare la frazione di 2 cifre a sinistra (due gruppi) e sottrarre 2 all'esponente. L'esponente è 69 - 64 = 5, e quello nuovo sarà quindi 3 (+64 = 67).

$$0\,1000011, 0001\ 1011\ 0000\ 0000 = 16^{3} \cdot (1 \cdot 16^{-1} + B \cdot 16^{-2}) = 432$$

Per l'esponente della frazione, prima converti ogni cifra (ogni gruppo in hexa, i singoli 1 e 0 in decimale, etc...) e moltiplicalo per la base elevato per la posizione che tale cifra occupa nella sequenza (al contrario) (in 1000000000000001 decimale, il primo uno sarà $1 \cdot 2^{-1}$, mentre l'ultimo sarà $1 \cdot 2^{-15}$).