

1. 浮點數加法

- step:
- 1. 對齊兩個 operand 的指數為一致 (指數小的向右移)
 - 2. 將 Significand 相加
 - 3. 將相加結果正規化, 確認是否有 overflow or underflow
 - 4. 將結果四捨五入

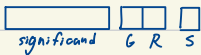
eg. $0.5_{(10)} + -0.4375_{(10)}$ 且為 4 個精確位數

$0.5_{(10)} \rightarrow 0.1_{(2)} \rightarrow 1.000 \cdot 2^{-1}$
 $-0.4375_{(10)} \rightarrow -0.0111 \rightarrow -1.110 \cdot 2^{-2}$

- step 1. 對齊指數 $= -1.110 \cdot 2^{-2} = -0.111 \cdot 2^{-1}$
step 2. significand 相加 $= \frac{1.000}{-0.111} = 0.001 \cdot 2^{-1}$
step 3. Normalize: $1.000 \cdot 2^{-4} = (1272-42-126)$
step 4. 四捨五入: $1.000 \cdot 2^{-4} = 0.0001 = 0.0625_{(10)}$

3. Rounding (四捨五入)

需要有 HW 支援, 即增加 guard bit 和 round bit



其中: G, R 為 guard bit 和 round bit 作為四捨五入的依據
S 為 sticky bit, 用作判斷運算過程中 GR 右邊是否有 bit 被 cut 掉

eg. $2.56 \times 10^0 + 2.34 \cdot 10^2$

0. without GR

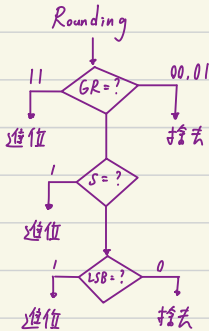
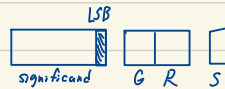
$$\begin{array}{r} 2.34 \times 10^2 \\ + 0.02 \cdot 10^2 \\ \hline 2.36 \cdot 10^2 \end{array}$$

0. with GR

$$\begin{array}{r} 2.34 \cdot 10^2 \\ + 0.0256 \cdot 10^2 \\ \hline 2.3656 \cdot 10^2 \\ = 2.37 \cdot 10^2 \end{array}$$

其中捨入方法為:

round to nearest even
即若 LSB 為奇時進位
偶捨去



判斷 Rounding 時需
考慮 LSB, G, R, S 四個 bit
判斷誤差大小可用 units in the last place (ulp)

units in the last place (ulp) The number of bits in error in the least significant bits of the significand between the actual number and the number that can be represented.

eg. $2.34 \cdot 10^5 + 2.56 \cdot 10^0$

actual number: $2.3400256 \cdot 10^5$

with GR: $2.34 \text{xxxx} \cdot 10^5$

ulp 越大表示誤差越多

要注意的是 ulp 為 * of bit, 而非差值!

2. 浮點數乘法

- step:
- 1. 將兩 operand 指數部份相加, 並扣除一次 bias
 - 2. 將 Significand 部份相乘
 - 3. Normalize 相乘結果, 並 check 是否 overflow 或 underflow
 - 4. 四捨五入乘積
 - 5. 設定乘積正負號, 若異號則為 1, 否則為 0

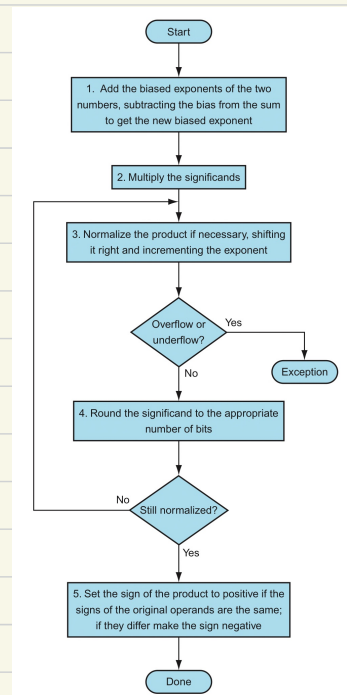
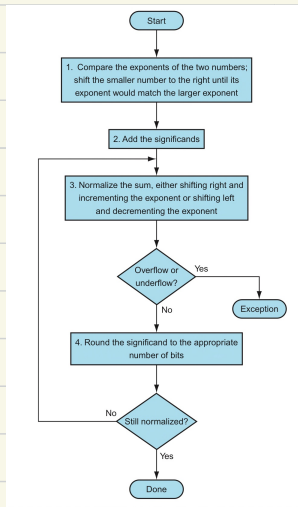
$\frac{S_1}{S_2} \cdot \frac{S_3}{S_4}$ 等同於 $S_1 \cdot S_3$ 和 $S_2 \cdot S_4$

eg. 若用 $0.5_{(10)} \times -0.4375_{(10)}$ 且為 4 個精確位數

$0.5 \rightarrow 1.000 \cdot 2^{-1}$
 $-0.4375 \rightarrow -1.110 \cdot 2^{-2}$

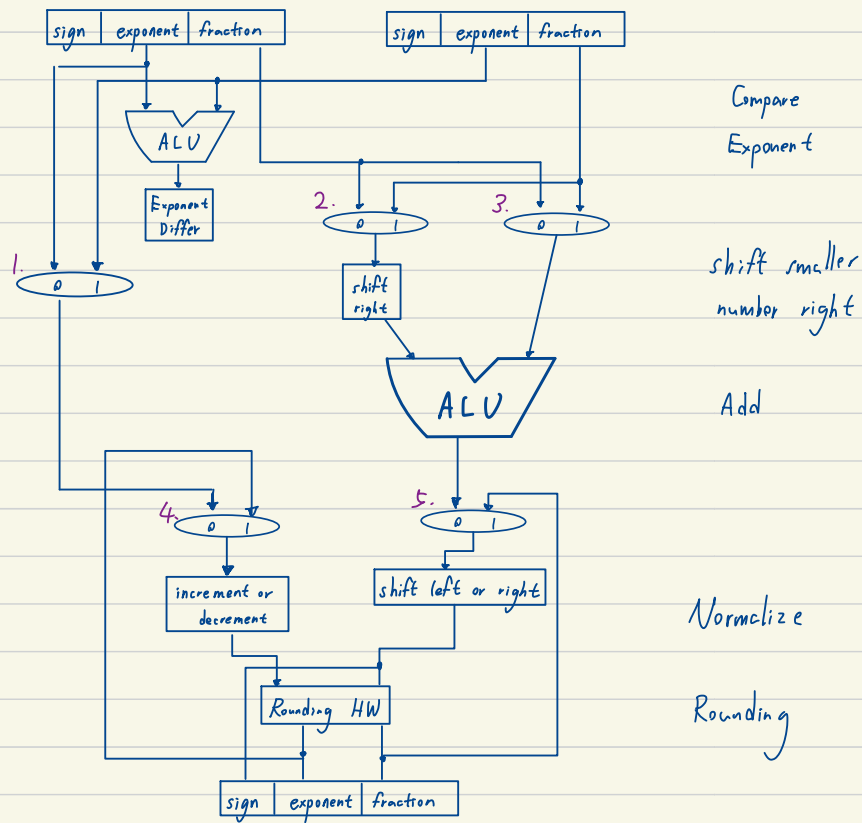
- step 1. 若以非 bias notation 計算: $-1 + -2 = -3$
以 bias notation 計算: $(-1+127) + (-2+127) - 127 = 124$

- step 2: $1.110 \times 1.000 = 1.110000 \cdot 2^{-3} = 1.110 \cdot 2^{-3}$
step 3: $1.110 \cdot 2^{-3}$ ($\because 127 - 3 \geq -126$)
step 4: $1.110 \cdot 2^{-3}$



補充:

浮點數加法硬體:



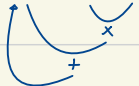
上圖中編號1~5的Mux用途為:

1. exponent 大的會作為加完結果之 exponent 傳下去
2. exponent 小的需調整至和大的相同
3. exponent 大的 fraction 原樣傳出
4. 是否為做完 rounding 後仍未 Normalized 的輸入
5. 同上.

⑩ Fused Multiply Add

源起：數位信號處理器上為了作傅利葉轉換，常需做乘加運算
為了加速，在 instruction set architecture 中會設計乘加指令

舉 multi-add \$t1, \$t2, \$t3



好處：
① instruction count 較低，performance 較好
② 當做乘再加時，會做又次捨入
而乘加指令中，只有乘完再加才做一次捨入
精準度較高

⑪ 右移和 = 進位除法關係

左移指令和 2 進位乘法相同，但右移和 2 進位除法不盡相同

舉 對有符號數不恆成立

-5₁₀ 除以 4₁₀，商數為 -1₁₀

-5：1011₂，右移後為：1110 = -2₁₀

可看出商數結果差 1， $\therefore -5 = 1011 = -2^3 + 2^1 + 2^0$
 $4 = 0100 = 2^2$

$$= -2^1 + 2^{-1} + 2^{-2} = -2 + 0.5 + 0.25 = -1.25 = -1$$

若為右移： $= 2^1 + 2^0 = 3$

可知，若右移捨去位元不為 0，則結果必差 1

解法：設為 I 要除以 8

若為正數 \Rightarrow 直接右移 $\Rightarrow I \gg 3$

若為負數 $\Rightarrow I + 7 \gg 3$

\therefore 若為：

$\begin{array}{r} 1000 \\ + 111 \\ \hline 1111 \\ \gg 3 \quad 111 \end{array}$	$\begin{array}{r} 1101 \\ + 111 \\ \hline 0100 \\ \gg 3 \quad 0000 \end{array}$
--	---

寫作 1 statement： $[I + (I \gg 3) \& 7] \gg 3$

\therefore int 為 4 byte = 32 bit

$\begin{array}{r} 1xxx \dots xx \\ \gg 31 \quad 1111 \dots 11 \\ \& 7 \quad 0000 \dots 111 \\ \hline 0000 \dots 111 \end{array}$	$\begin{array}{r} 0xx \dots xxx \\ \gg 31 \quad 010 \dots 010 \\ \& 7 \quad 000 \dots 111 \\ \hline 000 \dots 000 \end{array}$
--	--

⑫ 判斷是否有進位 設為無符號加法

舉 $1011 = \$t3$ $1011 + 0111 = 0010$
 $+ 0111 = \$t4$
 10010
 \therefore 加完結果必小於 \$t3, \$t4

設用 \$t4 來記錄 carry-out

\Rightarrow addu \$t2, \$t3, \$t4

sll \$t2, \$t2, \$t4 若 \$t4 > \$t2 \Rightarrow 有進位 \Rightarrow \$t2 = 1
\$t4 \leq \$t2 \Rightarrow 無 \Rightarrow \$t2 = 0

⑬ 浮點數加法結合律

浮點數加法不滿足結合律

\therefore 是以有限硬體表示實數，精準度會受限

舉 以 IEEE 754 單精度為例：

令 $x = -1.5 \times 10^{30}$ $y = 1.5 \times 10^{30}$ $z = 1.0$

則： $(x+y) + z = (-1.5 \times 10^{30} + 1.5 \times 10^{30}) + 1.0 = 0.0 + 1.0 = 1.0$

$x + (y+z) = -1.5 \times 10^{30} + (1.5 \times 10^{30} + 1.0 \times 10^0) = -1.5 \times 10^{30} + 1.5 \times 10^{30} = 0.0$

可知： $(x+y) + z \neq x + (y+z)$

⑭ 軟體偵測 overflow (100 交大) (97 清大) (105 中興)

iii. unsigned：邏輯：以 32 位元整數為例

(看 range) overflow 發生於： $\$t1 + \$t2 > 1111 \dots 1111 = 2^{32} - 1$

$$\Rightarrow \$t2 > (1111 \dots 1111) - \$t1 = 2^{32} - 1 - \$t1 = \$t1$$

Assembly code: addu \$t0, \$t1, \$t2

nor \$t3, \$t1, \$0 \ast \$t3 = not \$t1

sll \$t3, \$t3, \$t2 \ast if \$t2 > \$t1 then \$t3 = 1

bne \$t3, \$0, overflow

iv. signed：一般來說，使用 signed 運算指令 overflow OS 會判斷為 exception，發出 trap

\therefore unsigned 運算常用在記憶體位址運算上，overflow 用 mod 處理即可

邏輯：① 確認相加兩數是否同號，若否，不會 overflow

(看 sign) ② 確認兩數相加結果和兩數異-是否同號，若否，則 overflow

Assembly code: addu \$t0, \$t1, \$t2

xor \$t3, \$t1, \$t2

sll \$t3, \$t3, \$0 \ast 若 \$t1, \$t2 異號

bne \$t3, \$1, nooverflow \$t3 為 1

xor \$t3, \$t0, \$t1

sll \$t3, \$t3, \$0 \ast 若 \$t1+\$t2, \$t1 異號

bne \$t3, \$0, overflow \$t3 為 1

$$\begin{array}{r} 1xxx \dots xxx \\ \text{xor } 0xxx \dots xxx \\ \hline \Rightarrow 1xxx \dots xxx < 0 \end{array}$$