Progetto di reti Logiche

Sommatore Floating Point IEEE 754

Luca Cattani e Manuela Marenghi

Introduzione

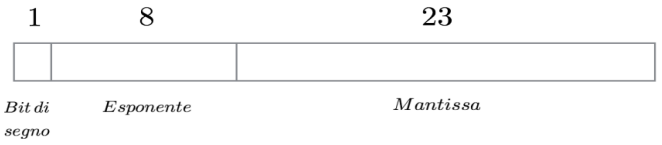
L'obiettivo principale di questo progetto è implementare un sommatore efficiente e preciso che sia in grado di eseguire l'operazione di somma tra due numeri in virgola mobile a singola precisione, rispettando lo standard IEEE 754. La precisione singola richiede la gestione accurata dei bit di segno, esponente e mantissa dei numeri, oltre all'implementazione di strategie di arrotondamento adeguate.

Queste regole prevedono la gestione di situazioni come l'overflow (quando il risultato supera la capacità massima rappresentabile) e l'underflow (quando il risultato è troppo piccolo per essere rappresentato).

Standard IEEE 754

Lo standard IEEE 754 è un insieme di regole e specifiche per la rappresentazione e l'aritmetica dei numeri in virgola mobile sui computer. Questo standard è stato sviluppato dall'Institute of Electrical and Electronics Engineers (IEEE) ed è ampiamente utilizzato nell'industria informatica per garantire una rappresentazione coerente dei numeri reali e delle operazioni matematiche su di essi.

In questo progetto viene utilizzata la rappresentazione a precisione singola, anche nota come "floating point a 32 bit", uno dei formati di numeri in virgola mobile definiti dallo standard IEEE 754. In questo formato, un numero in virgola mobile a singola precisione è rappresentato su 32 bit (4 byte). Con questa rappresentazione è possibile rappresentare numeri nell'intervallo che va da $1{,}18 \cdot 10^{-38}$ a $3{,}40 \cdot 10^{38}$ e i casi speciali ±0, ±∞, NaN ("not a number").



La rappresentazione di un numero in virgola mobile a singola precisione secondo lo standard IEEE 754 prevede tre componenti principali: il segno, l'esponente e la mantissa:

1. Il bit più significativo (posizione 31) è utilizzato per indicare se il numero è positivo o negativo.
2. Gli 8 bit successivi rappresentano l'esponente che viene calcolato sommando 127 all'effettivo esponente in modo da rendere confrontabili tutti i numeri rappresentabili in questo standard. Sia **e** l'esponente effettivo ed **E** l'esponente dello standard il calcolo avviene come segue:

$$E = e + 127$$

3. I restanti 23 bit rappresentano la mantissa ovvero la parte frazionaria del numero.

Detti S il segno, E l'esponente e M la mantissa la rappresentazione dei numeri secondo questo standard avviene seguendo la formula:

$$(-1)^s \cdot 1.M \cdot 2^E$$

Il primo 1 a sinistra della virgola viene omesso dalla mantissa perché sottinteso.

Sommatore a precisione singola

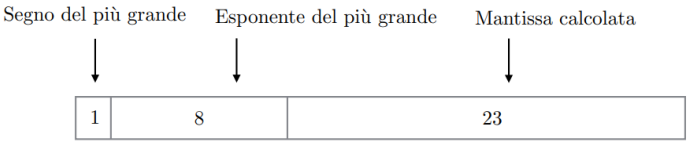
Per eseguire la somma di due numeri in virgola mobile a singola precisione, il sommatore IEEE 754 utilizza diversi passaggi. Innanzitutto, si verificano i segni dei numeri da sommare. Se i segni sono diversi, si esegue una sottrazione al posto di una somma. Successivamente si allineano gli esponenti dei numeri, in modo che abbiano lo stesso valore. A questo punto, si possono sommare le mantisse dei numeri allineati. Infine, si normalizza il risultato ottenuto per garantire la corretta rappresentazione in virgola mobile secondo lo standard IEEE 754.

Nella nostra implementazione oltre ad eseguire la somma di numeri con segno opposto è possibile, tramite un segnale in input, indicare se di due numeri si vuole eseguire somma o sottrazione (indipendentemente dal segno).

Oltre alle operazioni aritmetiche, lo standard IEEE 754 definisce anche regole per la gestione di casi speciali, come il trattamento di infinito, NaN (Not a Number) e zeri positivi e negativi.

L'algoritmo per sommare due numeri in virgola mobile è lo stesso sia per precisione singola che per precisione doppia. Supponendo di avere due numeri normalizzati in ingresso X e Y, i passi da seguire sono:

1. eseguire controlli riguardo alla presenza di casi speciali come NaN e infiniti nel caso in cui il risultato sia già predefinito;
2. cambiare il segno del secondo operando in caso di sottrazione;
3. identificare l'operando maggiore confrontando gli esponenti;
4. denormalizzare la mantissa dell'operando minore in base alla differenza tra i due esponenti (utilizzando uno shift a destra della differenza tra i due esponenti);
5. effettuare la somma dei due operandi (che sono già stati complementati in caso di segni diversi e di sottrazione);
6. si è ottenuto a questo punto il seguente:



7. normalizzare la mantissa a seconda della posizione dell' 1 più significativo e conseguentemente adeguato l'esponente;
8. controllare i segnali che indicano la presenza di casi speciali e vengono eseguiti gli arrotondamenti opportuni;

Gestione dei casi speciali

La rappresentazione dei casi speciali, ovvero ±0, ±∞, NaN , segue la tabella:

Categoria	Esponente	Mantissa
Zeri	0_{10}	0_{10}
Numeri denormalizzati	0_{10}	non zero
Numeri normalizzati	$1_{10} - 254_{10}$	qualunque
Infiniti	255_{10}	0_{10}
NaN	255_{10}	non zero

In questo progetto non tratteremo il caso dei numeri denormalizzati.

Per rappresentare risultati NaN abbiamo assunto come standard rappresentativo il seguente:

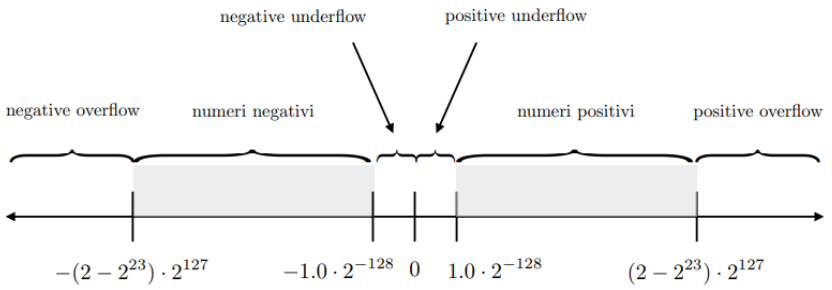
S11111111000000000000000000000001

In questo caso il segno (rappresentato da S) verrà assunto in maniera arbitraria.

Per eseguire la somma dei casi speciali è possibile invece seguire questa seconda tabella:

Operando 1	Operando 2		Risultato
Non speciale	Zero	=	Operando 1
Zero	Non speciale	=	Operando 2
NaN	–	=	NaN
–	NaN	=	NaN
± Infinito	∓ Infinito	=	NaN
± Infinito	± Infinito	=	± Infinito
± Infinito	Non speciale	=	± Infinito
Non speciale	± Infinito	=	± Infinito

Arrotondamenti



Per la gestione dei casi speciali abbiamo scelto i seguenti arrotondamenti:

- overflow, quando la somma di due numeri con lo stesso segno porta l'esponente a 255 (11111111) il risultato diventa ±inf in base al segno degli operandi;
- underflow, quando la differenza tra due numeri produrrebbe $-\ 2^{-128} < output < 2^{-128}$ si arrotonda a 0;
- Il comportamento con numeri denormalizzati non è definito;

Specifica

Il sommatore in questione è stato realizzato in modalità pipelined a tre stadi:

1. **Fase pre-somma**. In questa fase viene stabilito l'esponente più grande e conseguentemente la mantissa dell'operando minore viene denormalizzata tramite uno shifter a destra. In caso di sottrazione o di operandi negativi vengono fatte le opportune complementazioni. Sono inoltre individuati eventuali risultati speciali dovuti agli operandi speciali.
2. **Fase di somma**. Questa fase è determinata unicamente dalla somma effettiva delle mantisse e restituirà il risultato da normalizzare (in notazione unsigned) e il segno.
3. **Fase di normalizzazione e gestione casi speciali**. Rappresenta l'ultima fase, in cui la mantissa viene normalizzata con gli arrotondamenti necessari e l'esponente corretto. Viene inoltre gestita la presenza di operandi speciali, individuati nella prima fase.

Per gestire le tre fasi è stato scelto un ciclo di clock di 30 ns.

Fase pre-somma

Questa fase svolge in parallelo due funzioni:

1. Si calcola inizialmente il valore assoluto della differenza tra gli esponenti per conoscere quale dei due esponenti è più grande (e dovrà quindi essere preso come riferimento per il calcolo del risultato finale) e si shifta la mantissa del numero con l'esponente più piccolo in modo da poter fare, nello stadio successivo, la somma tra due numeri dello stesso ordine di grandezza. In questo modulo è calcolato, se necessario, il complemento a due delle mantisse dei due numeri anche in relazione al fatto che si debba operare una somma o una sottrazione.
2. Vengono analizzati la mantissa, il segno e l'esponente degli operandi in modo da capire se sono o meno dei numeri speciali (NaN o infiniti) in base allo standard IEEE 754, descritto nell'introduzione.

Fase di somma

In questa fase avviene l'effettiva somma tra le mantisse. Il componente scelto per effettuare questa somma è un RCA.

Durante questa fase il risultato viene esteso su 25 bit, utilizzando l'ultimo riporto della somma, per poter determinare nella fase successiva la normalizzazione correttamente e viene complementato il risultato in caso vi fosse necessità.

Fase di normalizzazione e gestione casi speciali

Quest'ultima fase è caratterizzata da due operazioni principali:

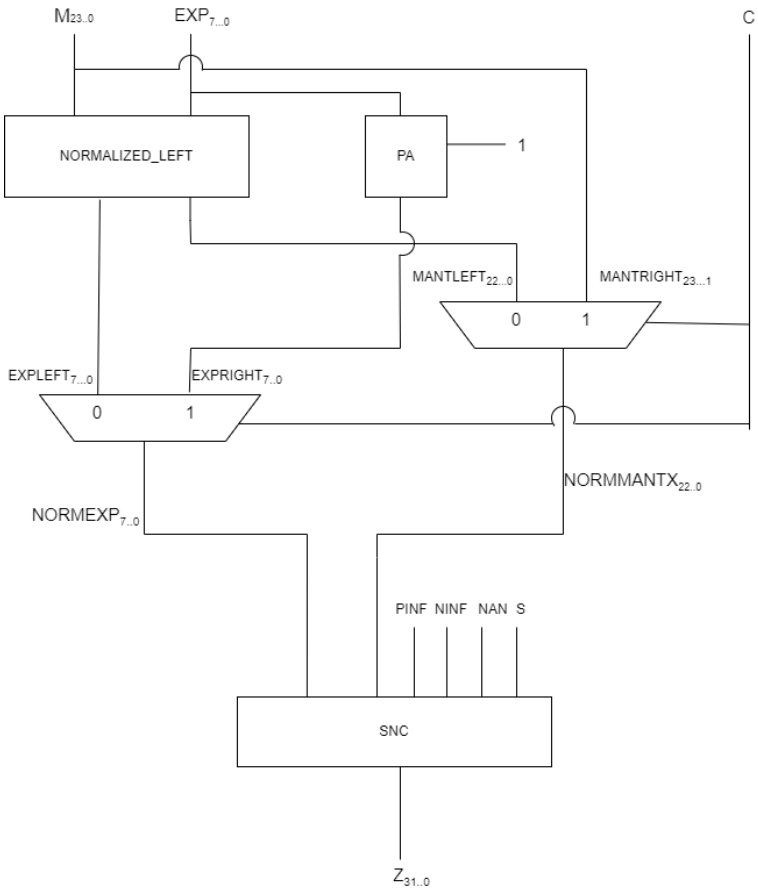
1. Shiftare a sinistra la mantissa del risultato e sottrarre all'esponente il numero di shift effettuati. Questo risultato sarà valido se l' 1 più significativo si trova a destra del 24-esimo bit.

   Dopodiché vengono effettuati dei controlli su mantissa ed esponente nel caso in cui debbano essere arrotondati a zero oppure alla rappresentazione di $\pm\infty$.

2. Sommare 1 all'esponente e considerare la mantissa a partire dal 24-esimo bit escludendo il bit meno significativo. In questo caso non è necessario fare ulteriori

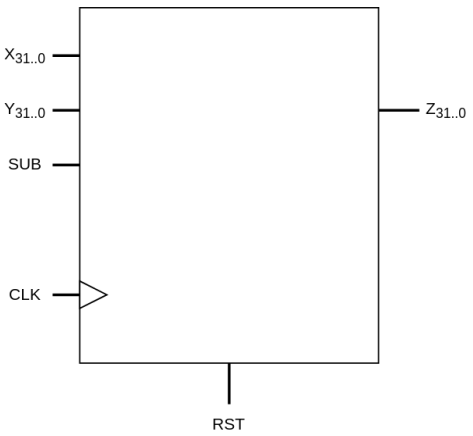
Questi due risultati vengono poi confrontati in base all'effettiva posizione dell'1 più significativo, stabilita dal 25-esimo bit della somma delle mantisse, secondo il quale viene propagata la soluzione corretta.

In seguito un altro componente si occupa di propagare la soluzione effettiva confrontando sia esponente che mantissa con i segnali speciali atti a indicare la presenza di NaN o infiniti. Vengono svolti anche eventuali arrotondamenti a zero o a infinito.



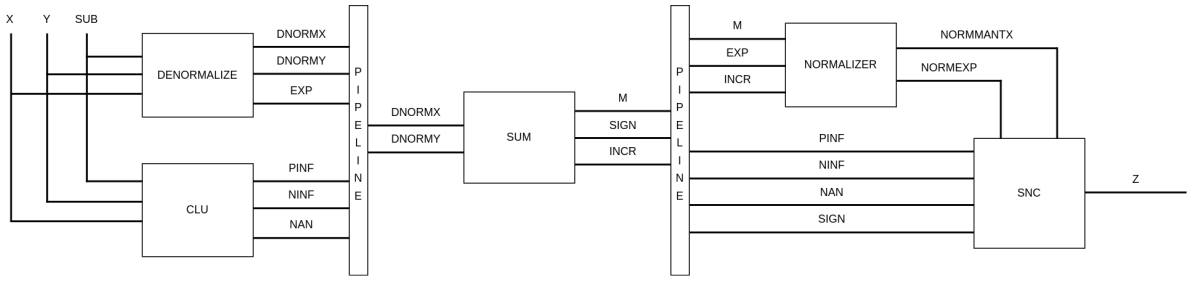
Interfaccia del sistema

Il sommatore prende in ingresso due operandi normalizzati X e Y e un segnale SUB che vale 1 se bisogna operare una sottrazione, 0 altrimenti. In uscita viene riportato un segnale normalizzato di 31 bit che rappresenta il valore dell'operazione svolta sui due ingressi. Sono inoltre presenti i segnali di reset RST e di clock CLK, è bene specificare che il dispositivo è sensibile al fronte di salita del segnale di clock.



Architettura del sistema

Il dispositivo utilizza una pipeline a tre stadi i cui ritardi sono stati stimati rispettivamente a 25.4 ns, 27.0 ns e 25.3 ns, per questo si è deciso di utilizzare un segnale di clock di 30 ns, di seguito una rappresentazione semplificata degli stadi della pipeline:



Il primo stadio prende gli input del top-level produce i numeri X e Y normalizzati, DNORMX corrisponde sempre al numero più piccolo e DNORMY al numero più grande, nel caso il numero più piccolo sia negativo sarà prodotto il suo complemento a uno (a cui verrà sommato 1 nel prossimo stadio della pipeline per produrre il complemento a due), il numero più grande viene invece eventualmente inoltrato in complemento a due. Gli input del top-level sono inoltre analizzati dal modulo CLU per produrre i segnali PINF, NINF e NAN (tutti da un solo bit) che indicano se l'output dell'operazione è un numero speciale, è importante notare che questi segnali sono one-hot, non è infatti possibile che il risultato possa essere allo stesso tempo due numeri speciali diversi. Questi segnali possono valere 1 solo se almeno uno degli input del top-level è un numero speciale. Viene anche inoltrato allo stadio successivo l'esponente del numero più grande.

Il secondo stadio si occupa di eseguire la somma tra le mantisse dei numeri denormalizzati e di produrre un segnale su 24 bit che rappresenta la mantissa denormalizzata (comprensiva dell'1 omesso nella rappresentazione normalizzata) in notazione unsigned. Vengono poi inoltrati allo stadio successivo il segnale SIGN da un solo bit dal valore 1 se il risultato dell'operazione è negativo, 0 altrimenti (non è previsto un valore fisso per SIGN nel caso il risultato sia zero) e il segnale INCR da un bit che indica se a causa di un overflow nella somma sarà necessario incrementare l'esponente del risultato. I segnali non utilizzati sono inoltrati allo stadio successivo.

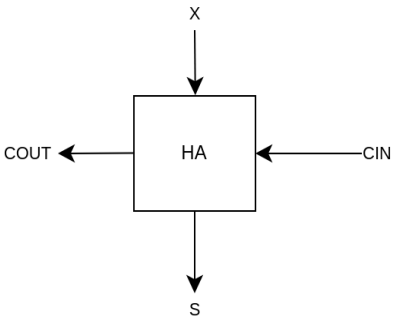
Il terzo stadio si occupa in primo luogo di normalizzare il risultato ottenuto allo stadio precedente shiftando la mantissa M a destra o a sinistra in base all'output di un priority encoder e sottraendo al segnale EXP il numero di bit di cui la mantissa viene shiftata verso destra. Dopo aver ottenuto un numero normalizzato si controlla in base ai valori dei segnali PINF, NINF e NAN se il risultato deve essere modificato a causa di un operando speciale. Il segnale SIGN viene copiato nel bit più significativo del risultato per indicare il segno.

È bene ricordare che gli ingressi e le uscite sono campionate ogni 30 ns, l'output corretto sarà quindi disponibile solo dopo quattro cicli di clock.

Moduli Base

In seguito vengono descritti i moduli basilari utilizzati per realizzare la maggior parte dei componenti.

HA (Half Adder)

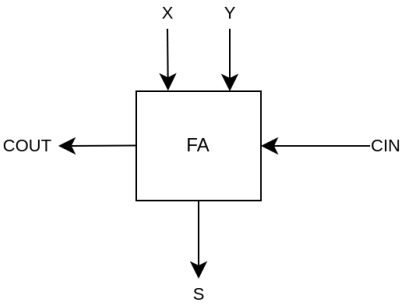


L' half adder si occupa di sommare un bit X ed un riporto CIN, riportando il risultato della somma al segnale di uscita S e il riporto della somma sul segnale di uscita COUT. Le uscite sono governate dalle seguenti relazioni:

$$S = X\ xor\ CIN$$

$$COUT = X\ and\ CIN$$

FA (Full Adder)

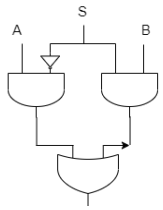


Il full adder si occupa di sommare due bit X e Y ed un riporto CIN, riportando il risultato della somma al segnale di uscita S e il riporto della somma sul segnale di uscita COUT. Le uscite sono governate dalle seguenti relazioni:

$$S = X\ xor\ Y\ xor\ CIN$$

$$COUT = (X\ and\ Y)\ or\ (Y\ and\ CIN)\ or\ (X\ and\ CIN)$$

MUX (multiplexer)



Un multiplexer, spesso abbreviato come MUX, è un circuito digitale che consente di selezionare uno tra diversi segnali di input e instradare quel segnale selezionato all'uscita. Il numero di segnali di input selezionabili dipende dal tipo di multiplexer. A caratterizzare il funzionamento di un mux con due bit A e B in ingresso e un segnale S è la seguente formula:

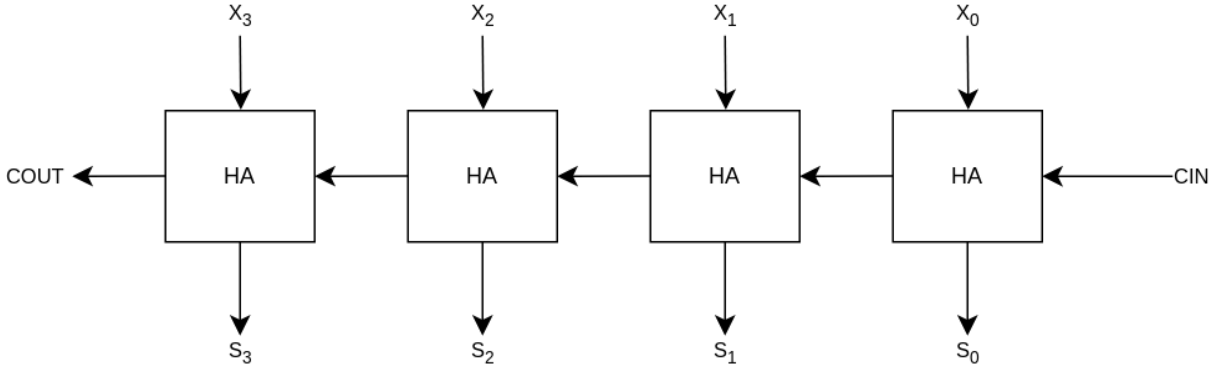
$$Y = A\ and\ (\ not\ S\ )\ +\ B\ and\ S.$$

In questo caso se S = 0 verrà propagato il valore di A, altrimenti il valore di B.

È possibile realizzare un mux su più bit semplicemente concatenando più mux su due bit.

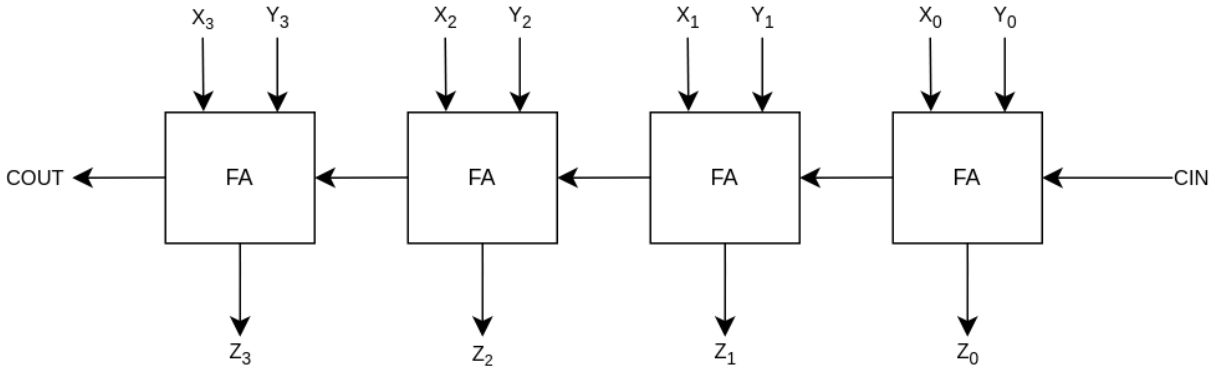
PA (Partial Adder)

Il sommatore parziale prende in ingresso una stringa di bit X e somma l'ingresso CIN tramite una serie di half adder (HA) che producono ognuno il risultato della somma tra un bit e il riporto della somma precedente, il modulo produce l'output S, che rappresenta il risultato della somma tra X e CIN, e COUT, che rappresenta l'eventuale riporto dell'operazione.



RCA (Ripple Carry Adder)

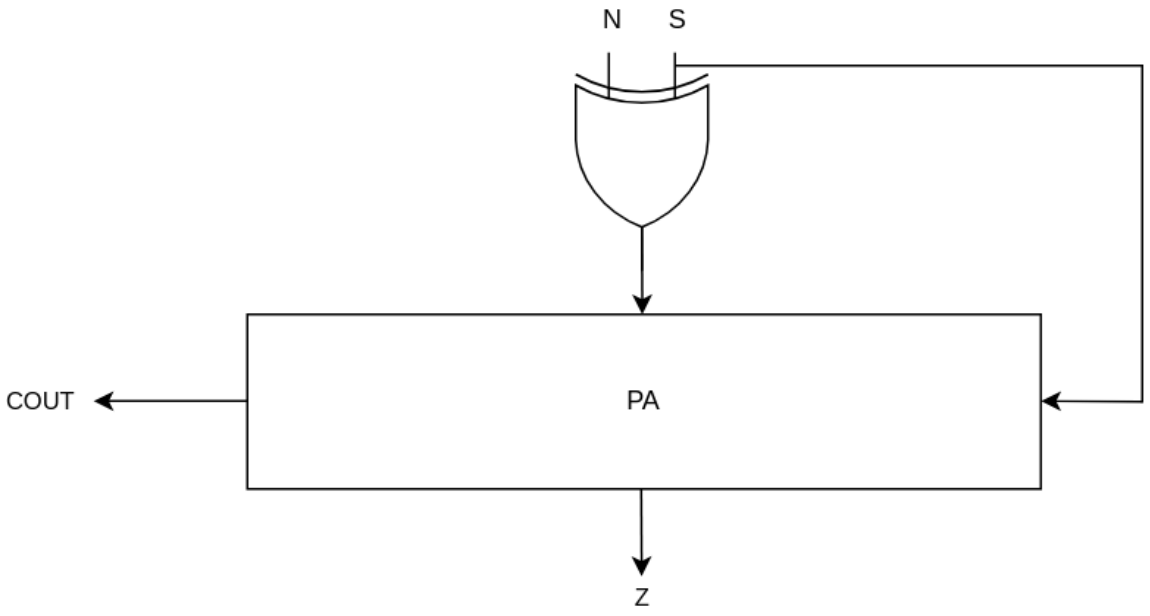
Il ripple carry adder è utilizzato per operare la somma tra due stringhe di bit di uguale lunghezza X e Y e un eventuale riporto iniziale da un bit CIN, la struttura interna del componente è costituita da una serie di full adder che si occupano di operare la somma tra due soli bit alla volta e un riporto, ogni full adder contribuisce al risultato finale propagando al full adder successivo il riporto della somma e producendo uno dei bit dell'uscita Z, il modulo produce anche un uscita COUT che rappresenta l'eventuale riporto prodotto dall'ultima somma. Nei moduli più complicati sono stati utilizzati RCA di lunghezza diversa, la struttura generale è comunque rappresentata di seguito.



C2C (Conditional 2's Complement)

Il complemento a due condizionale prende in ingresso la stringa di bit N da elaborare e si occupa di convertirla al suo complemento a due quando l'ingresso S vale 1, per fare ciò utilizza un Partial Adder (PA) a cui viene sottoposto l'ingresso $N\ xor\ S$, viene inoltre portato il segnale S all'ingresso CIN di PA per ottenere eventualmente il complemento a due di N. L'output di PA è poi riportato alle porte di uscita del modulo C2C. Osservando la struttura del modulo, si distinguono due casistiche:

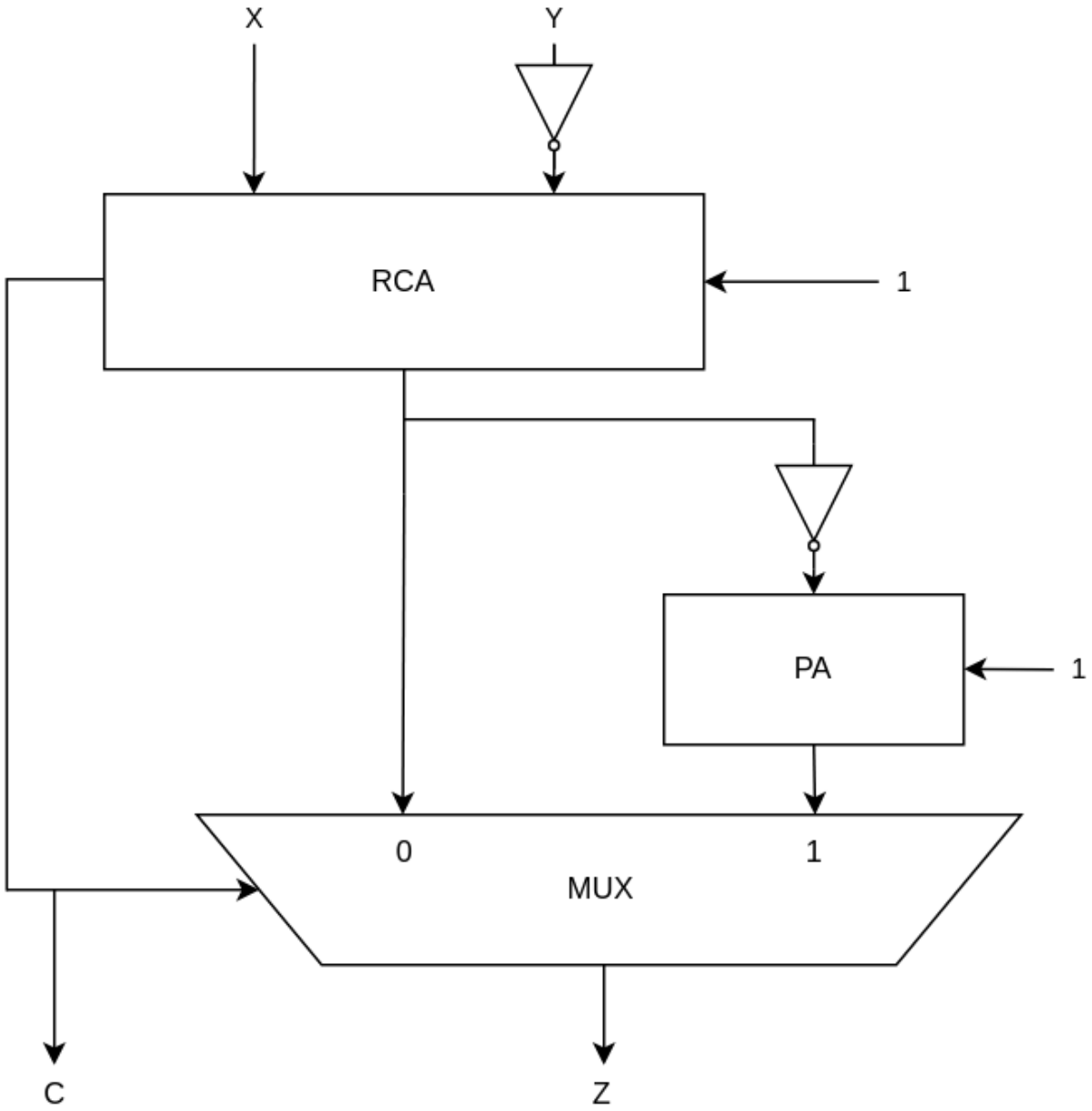
- S = 0 →PA avrà come input N e 0, effettuerà quindi la somma tra N e 0 producendo un'uscita proprio uguale ad N, l'uscita Z del modulo C2C sarà quindi uguale all'ingresso.
- S = 1 →PA avrà come input l'opposto di N e 1, effettuerà quindi la somma tra il complemento a uno di N e 1, ottenendo il complemento a 2, questo risultato verrà riportato all'uscita Z del modulo C2C.



ABSDIFF (Absolute Difference)

Questo modulo calcola il valore assoluto della differenza tra due numeri, per fare ciò si utilizza inizialmente un RCA per calcolare la differenza tra gli ingressi X e Y, fornendo al sommatore il complemento a uno di Y e impostando il suo ingresso CIN a 1. Dopo aver effettuato la sottrazione tra X e Y si ottiene dall'overflow della somma (uscita COUT del sommatore) quale dei due numeri è il più grande, questo segnale vale 1 se Y è maggiore di X, 0 altrimenti ed è portato all'uscita C del modulo. Dopo aver calcolato la differenza tra gli ingressi, si utilizza un modulo PA che, portando al suo ingresso CIN 1 e al suo ingresso X il risultato del sommatore negato, calcola il complemento a due della somma. A questo punto si ha il risultato del sommatore sia positivo che negativo, si distinguono quindi due casi:

- C = 0 → dal sommatore si è ottenuto che X > Y, quindi il risultato del sommatore corrisponde al modulo della differenza, allora il multiplexer porta questo segnale all'uscita.
- C = 1 → dal sommatore si è ottenuto che Y > X, quindi il risultato del sommatore era negativo, allora il modulo della sottrazione è stato calcolato dal partial adder.

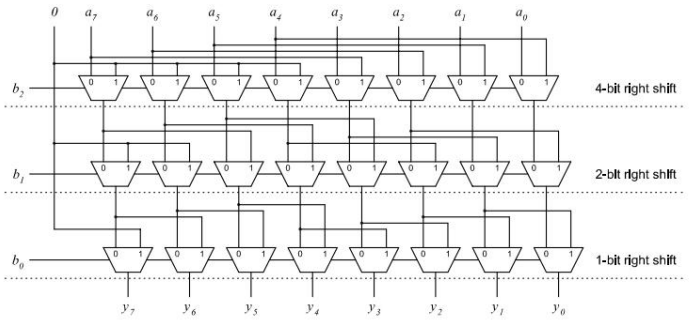


SHIFTER (RIGHT or LEFT)

Questo modulo si occupa di shiftare un segnale in input (nel nostro caso si tratta sempre della mantissa, quindi su 23 bit) di un numero di posizioni definito da un numero binario in input. Per realizzare questo componente si è deciso di utilizzare una soluzione logaritmica, ovvero effettuare gli shift in base al logaritmo della posizione del bit nel numero di shift.

In questo modo si effettuano shift di 2,4,8… posizioni in base al valore del bit del numero di shift da effettuare scomponendolo così in somme di potenze di 2 (questo solo se il bit in questione ha valore 1).

Questo tipo di shifter è anche detto barrel shifter e nel nostro caso prende il segnale del numero di shift da effettuare su 8 bit, utilizzando però la tecnica logaritmica solo per i 5 bit meno significativi (se almeno uno degli altri tre bit ha valore 1 allora si ha che il risultato sarà una serie di 0).



In figura viene riportato un esempio di barrel shifter con tre livelli di mux a titolo esemplificativo.

PRIORITY ENCODER

Il priority encoder è un circuito digitale utilizzato per assegnare una priorità a un insieme di segnali di input binari. Questo circuito riceve più segnali di input e determina quale di essi ha la priorità più alta, assegnando un codice di output in base a tale priorità.

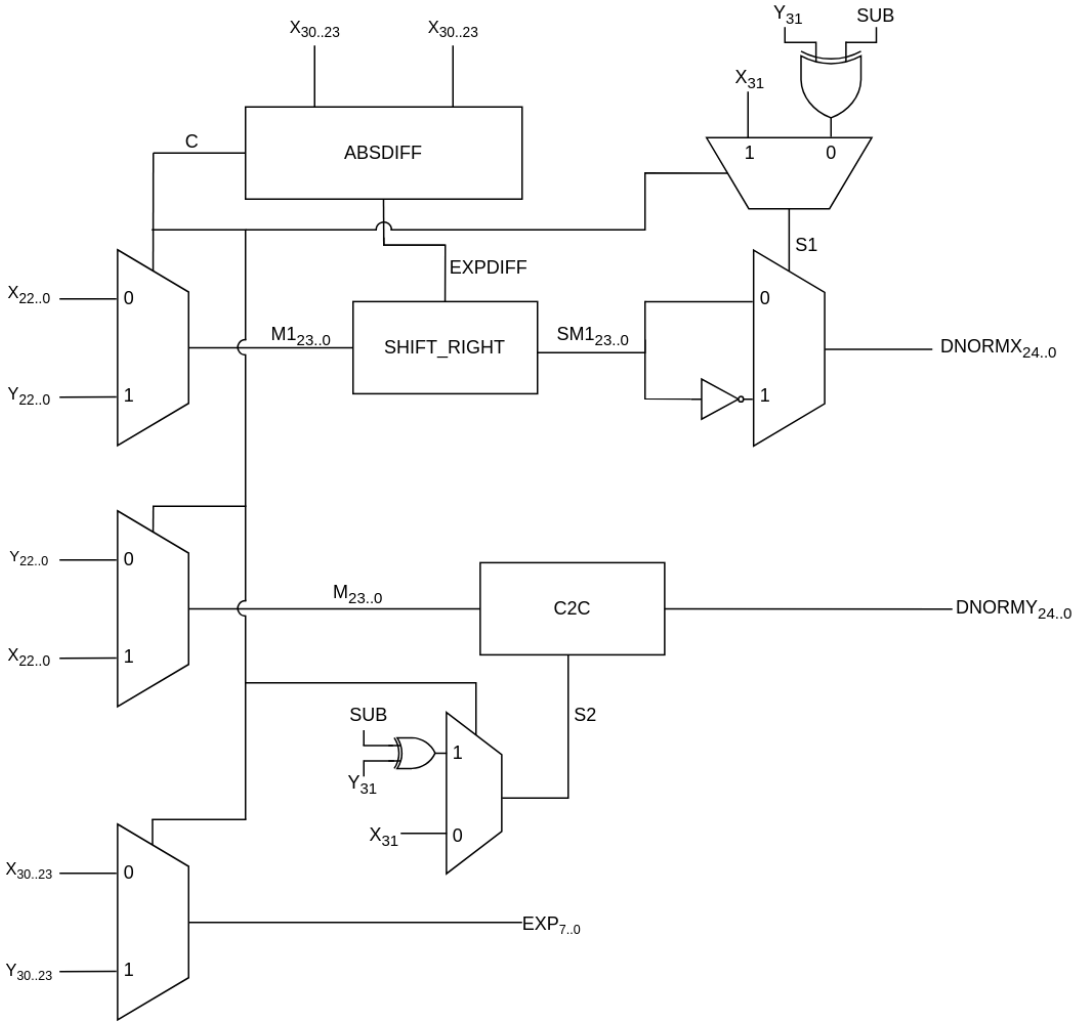
Il codificatore prioritario funziona analizzando i segnali di input in sequenza e identificando il primo segnale che è attivo (con valore "1"). Una volta trovato il segnale attivo con la priorità più alta, il codificatore assegna un codice binario di output che rappresenta la posizione di quel segnale nel suo ordine di priorità.

Nel progetto in questione ha la funzione di restituire il numero di shift da effettuare in base alla posizione dell'1 più significativo a destra della virgola.

Fase di pre-somma

DENORMALIZE (denormalizzatore)

Il denormalizzatore si occupa di propagare nei registri della pipeline l'esponente dell'operando più grande e di fornire allo stadio successivo della pipeline due mantisse di un ordine di grandezza comparabile per eseguire la somma.



In primo luogo è necessario calcolare il modulo della differenza dei due esponenti, per questo si utilizza un modulo ABSDIFF, le sue uscite sono riportate nei segnali interni EXPDIFF (vettore di otto bit che rappresenta il modulo della differenza) e C, segnale da un bit che assume il valore 1 se l'esponente di Y grande di quello di X, 0 altrimenti.Successivamente è necessario svolgere operazioni differenti sulle mantisse del numero più grande e di quello più piccolo (comprensive dell'1 omesso dalla notazione IEEE 754) e sugli esponenti:

- **Esponenti.** il segnale intermedio C viene utilizzato per selezionarel'esponente dell'operando più grande, il risultato viene inoltrato ai registri della pipeline tramite l'uscita EXP.
- **Numero più grande.** tramite un multiplexer viene selezionata la mantissa del numero più grande e inoltrato ad un modulo C2C che si occupa di convertirlo in complemento a due se necessario, il risultato è inoltrato allo stadio successivo tramite l'uscita DNORMY.
- **Numero più piccolo.** tramite un multiplexer viene selezionata la mantissa del numero più piccolo, che segnale deve essere shiftata a destra per un numero di volte pari alla differenza in valore assoluto degli esponenti (rappresentato del segnale interno EXPDIFF), per fare ciò è

utilizzato un barrel shifter. Dopo aver shiftato la mantissa, il risultato è complementato a uno se il numero è negativo e inoltrato al prossimo stadio tramite l'uscita DNORMX.

Per la conversione in complementa a uno o a due delle mantisse si considerano il primo bit dell'operando X (0 se positivo, 1 se negativo) e il risultato di uno xor tra il segno del'operando Y e l'ingresso SUB del top-level che indica se l'operazione da trattare è una somma o una sottrazione.

CLU (Control Logic Unit)

L'unità di controllo logico si occupa di gestire i casi particolari che si possono verificare eseguendo la somma con i numeri speciali NaN e infinito (positivo e negativo). Il modulo si trova nella prima fase della pipeline e utilizza come ingressi i due numeri da sommare (X e Y) nella loro rappresentazione secondo lo standard IEEE 754 e il bit che indica se l'operazione da trattare è una somma o una sottrazione (SUB), il modulo ha tre uscite: PINF (per indicare se il risultato è un infinito positivo), NINF (per indicare se il risultato è un infinito negativo) e NAN (per indicare se il risultato non è un numero).

Si ricorda lo standard IEEE 754 per NaN e infiniti positivi e negativi:

- Nan ⟶ segno qualunque, esponente = 255, mantissa diversa da 0
- Infinito positivo ⟶ segno = 0, esponente = 255, mantissa = 0
- Infinito negativo ⟶ segno =1, esponente = 255, mantissa = 0

Il modulo utilizza due segnali interni da due bit (XINF e YINF) in cui il primo bit indica il segno del numero e il secondo bit indica se si tratta di un infinito o meno, si distinguono 4 casi:

- 00 ⟶ numero finito positivo
- 01 ⟶ infinito positivo
- 10 ⟶ numero finito negativo
- 11 ⟶ infinito negativo

Vengono utilizzati anche altri due segnali interni (XNAN e YNAN) per indicare se i numeri sono NAN o meno, questi segnali valgono 1 quando l'esponente è 255 e la mantissa non 0.

Dopo aver controllato se X o Y rappresentano un numero speciale, e considerando la sottrazione come una normale somma con cambio di segno all'addendo di destra, si verifica il soddisfacimento delle seguenti condizioni.

L'implementazione del modulo può essere rappresentata secondo il seguente pseudocodice:

```
if (X.isNaN || Y.isNaN) return NaN

else if (X.isInfinite){
    if(Y.isFinite) return X
    else if (haveOppositeSign(X, Y)) return NaN
    else return X
}

else if (Y.isInfinite) return Y

return null      //non ci sono addendi speciali
```

si noti che le funzioni utilizzate sopra verificano se gli addendi sono numeri speciali seguendo le regole della notazione IEEE 754, ovvero confrontando segno, esponente e mantissa degli ingressi.

Nan

La somma tra due numeri produce NaN nel caso in cui almeno uno tra gli addendi sia a sua volta NaN oppure se si cerca di effettuare la somma tra infiniti di segno diverso. Ciò significa che si produce 1 se uno tra XNAN e YNAN è uno o se i primi bit di XINF e YINF sono diversi e i secondi bit sono entrambi 1 (uno deve valere 11 e l'altro 01).

Infinito

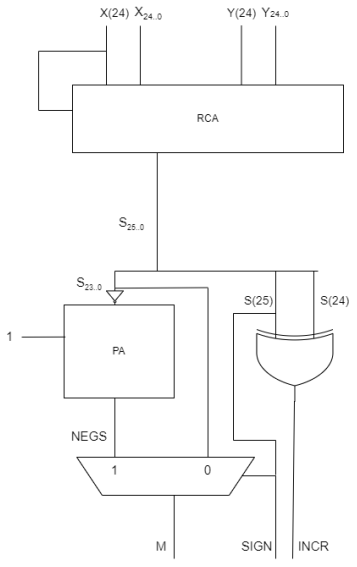
La somma tra due numeri produce un infinito positivo solo nei casi in cui si sommino: due infiniti positivi oppure un infinito positivo ed un numero finito qualunque (o viceversa). Analogamente si produce un finito negativo se si sommano due infiniti negativi o un solo infinito negativo e un numero non infinito. É importante osservare che si possono ottenere infiniti anche da operazioni apparentemente valide, ovvero dalla somma di due numeri non speciali. Quando i due numeri sommati sono molto grandi infatti si produrrebbe un risultato troppo grande (positivo o negativo) per poterlo rappresentare con la notazione IEEE 754, il risultato ottenuto quindi deve essere un infinito (positivo o negativo), questo caso non è gestito in questo modulo.

Fase di somma

Questa fase effettua la somma tra le due mantisse che, precedentemente adeguate al calcolo, vengono ulteriormente modificate estendendole di un bit a sinistra (duplicando il segno) per ricavare un bit di buffer che servirà per ottenere l'uscita INCR. Questo bit in più infatti assumerà nel risultato il valore 1 se l'esponente deve essere incrementato, 0 altrimenti.

Considerando il numero più grande (per semplicità, dato che la mantissa di 23 bit non viene modificata) la rappresentazione assunta in questa fase dai segnali prevede (da destra a sinistra):

- 23 bit per la mantissa come rappresentati in forma normalizzata
- 1 bit dal valore 1 che è omesso nella rappresentazione normalizzata
- 1 bit di buffer (dal valore di 1 se il numero è negativo, 0 se positivo)
- 1 bit che rappresenta il segno del numero

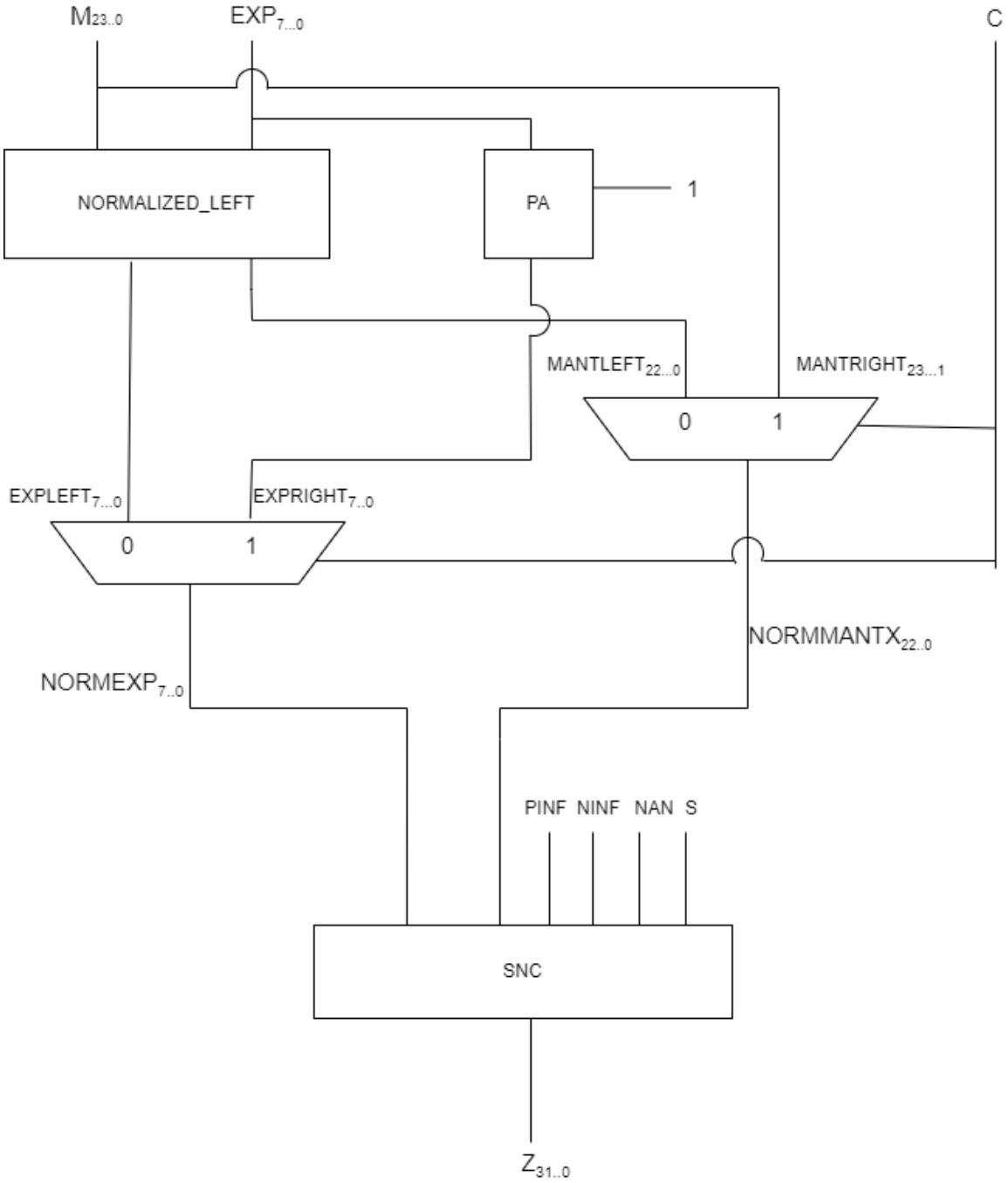


Per fare la somma si è deciso di utilizzare un RCA su 26 bit. Si noti che il sommatore prende in ingresso anche il bit di segno dell'operando più piccolo alla porta CIN. Ciò è dovuto al fatto che l'operando più piccolo è fornito al modulo nella sua forma in complemento a uno,quindi sommare il bit di segno alla porta CIN significa effettuare il complemento a due, se necessario. Infatti si somma 1 se il numero è in complemento a uno (ottenendo il complemento a due), 0 se il numero è positivo. Per poter svolgere l'intera fase sono stati utilizzati due segnali intermedi:

- **S** ovvero il risultato della somma (uscita del ripple carry adder) su 26 bit.
- **NEGS** per rappresentare il risultato (che corrisponde al segnale S) complementato a due da un partial adder.

Un multiplexer si occupa di portare il segnale corretto tra S e NEGS all'uscita del componente utilizzando il bit di segno di S, se S ha segno negativo si prende NEGS, altrimenti si prende S per avere la mantissa positiva.

Fase di normalizzazione e gestione casi speciali



NORMALIZER

Questo componente si occupa di normalizzare il risultato della somma tra le mantisse e di fare opportuni arrotondamenti per mantissa e segno. Nello specifico è composto a sua volta da i seguenti componenti:

- **NORMALIZED_LEFT**. Questo componente, che verrà descritto nello specifico in seguito, si occupa di shiftare la mantissa a sinistra fino a normalizzarla e di decrementare adeguatamente l'esponente. In caso il risultato fosse 0 o underflow (non rappresentabile) esegue l'arrotondamento a 0. Il risultato di questo modulo viene propagato dai segnali intermedi MANTLEFT ed EXPLEFT.

- **PA.** Questo componente prende in input l'esponente (segnale EXP) e lo incrementa di uno propagando il risultato tramite il segnale EXPRIGHT.
- **MUX1.** Il mux in questione prende in input MANTLEFT ed MANTRIGHT e propaga il segnale in base al valore del 25-esimo bit, ovvero il segnale in input C.
- **MUX2.** In questo caso ad essere presi in input saranno i due esponenti calcolati, ovvero EXPLEFT ed EXPRIGHT, e come segnale determinante sempre il 25-esimo bit (ovvero C).

L' obiettivo di questo componente è di normalizzare la mantissa, ovvero di far in modo che il 24-esimo bit (che viene omesso) sia l'1 più significativo e che l'esponente sia coerente con le modifiche effettuate alla mantissa (si potrebbe pensare come all'ordine di grandezza dell'1 più significativo).

Per farlo si possono presentare solo due casi:

1. 1 più significativo a sinistra della virgola, per cui al 25-esimo posto. In questo caso basta incrementare di 1 l'esponente per renderlo coerente e propagare la mantissa partendo dal 24-esimo bit fino al primo, ovvero operando uno shift di 1 a destra. In questo modo si ottengono i segnali intermedi MANTRIGHT e EXPRIGHT.
2. 1 più significativo a destra della virgola per cui è necessario operare lo shift a sinistra di tante posizione quanti gli zeri tra il primo 1 e la virgola. Viene quindi poi decrementato l'esponente del numero di shift effettuati. In questo modo si ottengono i segnali intermedi MANTLEFT ed EXPLEFT in uscita dal componente NORMALIZED_LEFT.

Nel caso in cui il risultato fosse già normalizzato (ovvero il bit più significativo fosse al 24 posto) lo shift effettuato sarebbe di zero posizioni.

La propagazione del risultato corretto è effettuata da un multiplexer in base all'ingresso C (a cui viene portato il segnale INCR):

1. C = 1. I risultati corretti saranno MANTRIGHT ed EXPRIGHT.
2. C = 0. I risultati corretti saranno MANTLEFT ed EXPLEFT.

NORMALIZED_LEFT

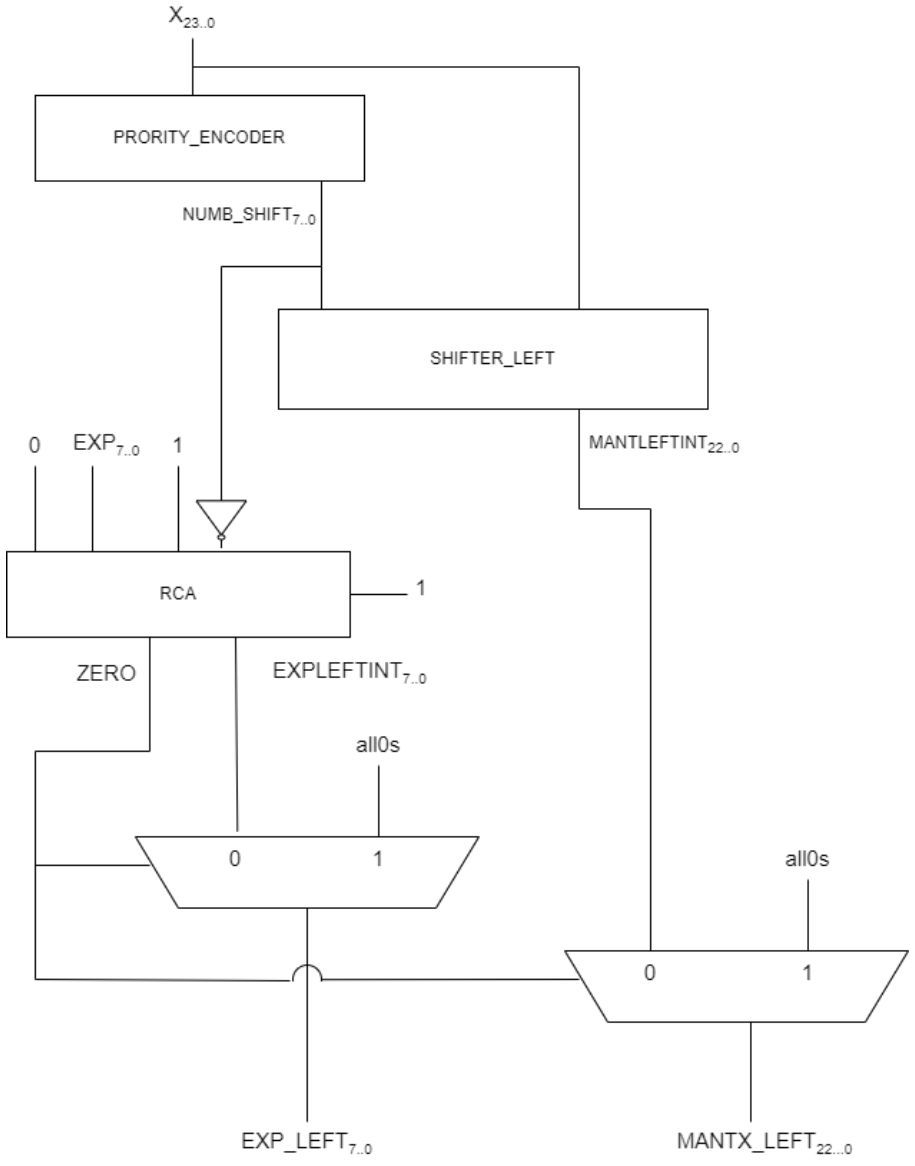
Questo componente si occupa di calcolare i segnali intermedi MANTLEFT ed EXPLEFT relativi al calcolo di mantissa ed esponente quando il 25-esimo bit assume valore nullo. I componenti principali risultano essere:

- **PRIORITY ENCODER.** Questo componente prende in input la mantissa risultante dalla fase precedente di pipeline e restituisce il numero di shift da effettuare a sinistra, in base alla posizione del bit più significativo a destra della virgola. In caso di mantissa composta da soli zeri verrà restituita la cifra "11111111". In seguito la situazione verrà trattata a dovere.Questo risultato viene propagato dal segnale intermedio NUMB_SHIFT.
- **SHIFTER_LEFT.** Si tratta di uno shifter di n bit a sinistra. Prende in input un numero su 23 bit e il numero di shift da eseguire (numero a 8 bit) e propaga il risultato tramite il segnale MANTLEFTINT. Per realizzare questo shifter si è scelta la versione logaritmica, anche detta barrel shifter.
- **RCA.** Si tratta di un ripple carry adder su 9 bit che prende in input EXP (esteso a 9 bit ponendo uno 0 in MSB) e vi somma il complemento a uno del segnale NUMB_SHIFT (sempre esteso a 9 bit ma aggiungendo un 1 trattandosi di un numero negativo). Viene quindi posto CIN => 1 per eseguire il complemento a due di NUMB_SHIFT. In questo modo si ottiene un risultato su 9 bit di cui i primi 8 bit definiscono il segnale intermedio EXPLEFTINT mentre il bit

più significativo viene propagato come segnale intermedio ZERO (se il risultato della sottrazione è negativo infatti significa che il modulo del numero è minore di 2^{-128}, quindi viene approssimato a zero).

- **MUX1.** Questo componente prende in input EXPLEFTINT e un segnale all0s e in base al valore del segnale intermedio ZERO propaga l'esponente corretto. Questo passaggio è necessario nel caso vi siano underflow, quindi si cerchi di sottrarre all'esponente un numero maggiore di se stesso per cui si effettua un arrotondamento a 0.
- **MUX2.** Analogamente si effettua lo stesso controllo sulla mantissa, per cui si prendono in input MANTLEFTINT e all0s e si propaga in base al valore del segnale ZERO in modo da adottare gli arrotondamenti opportuni.

Le descrizioni dei singoli componenti spiegano già le operazioni svolte dal componente principale per cui basti precisare che il funzionamento generale è quello di shiftare la mantissa a sinistra di un numero di posizioni determinate dal priority encoder e di sottrarre lo stesso numero all'esponente. In caso si siano verificati underflow durante questa operazione una coppia di mux esegue l'arrotondamento di mantissa ed esponente a 0.



SNC (Special Numbers Controller)

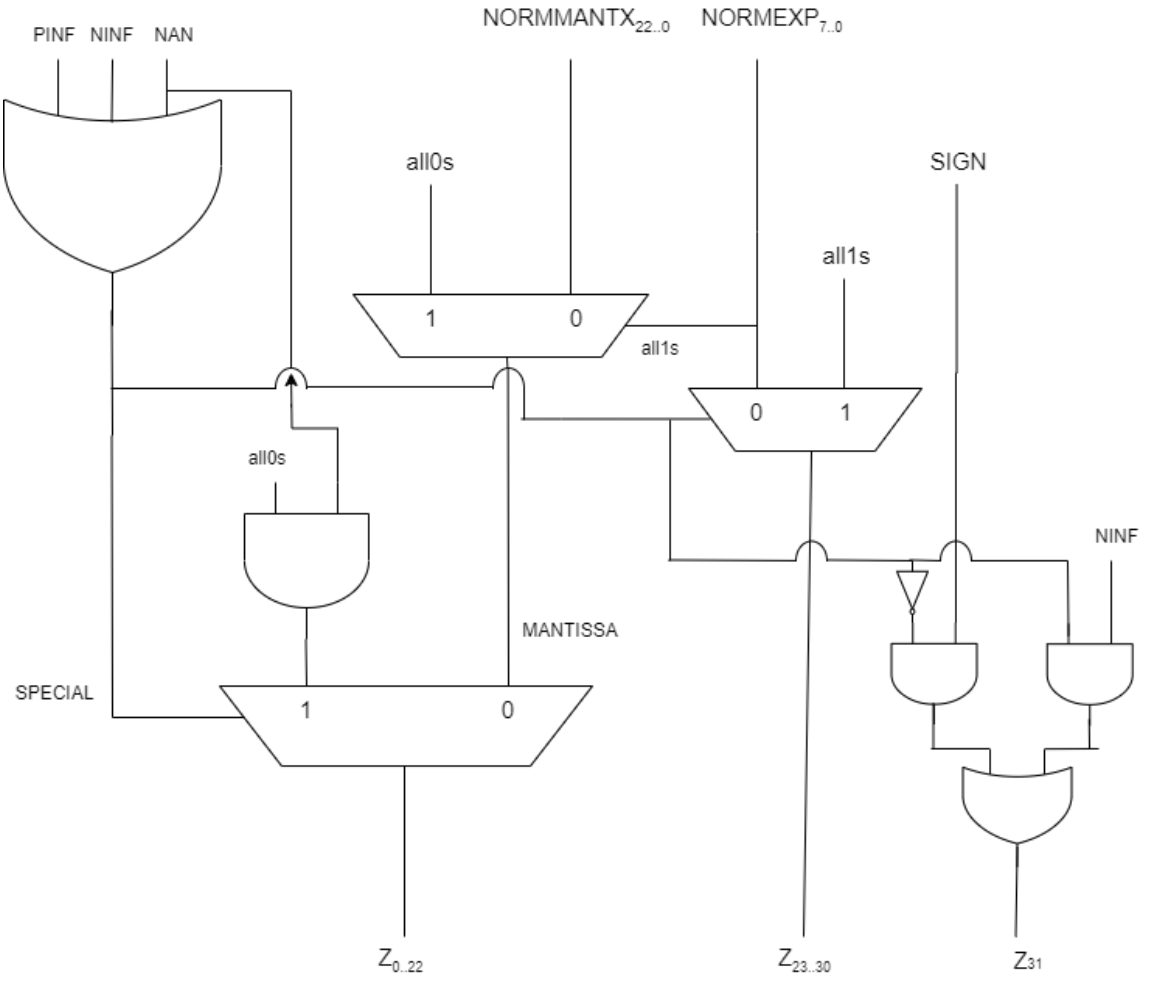
Questo componente si occupa di eseguire gli arrotondamenti finali e di gestire i casi speciali precedentemente definiti. Per farlo si utilizzano due segnali intermedi:

- **SPECIAL** che assume valore 1 se almeno uno dei tre segnali in ingresso (PINF, NINF e NAN) è 1, 0 altrimenti.
- **MANTISSA** che diventa tutti 0 se l' EXP in ingresso è una sequenza di otto 1 e quindi il risultato sarà infinito.

Di fatto vengono eseguiti controlli sulle tre parti risultate dalle fasi precedenti:

- **SEGNO.** Si stabilisce il segno in base al valore del segnale SPECIAL e del segnale NINF. Infatti il segno risultante viene propagato senza ulteriori controlli a meno del caso di -inf in cui è importante accertarsi che il segno sia negativo.
- **ESPONENTE.** L 'esponente subisce modifiche solo qualora vi siano casi speciali (quindi SPECIAL = 1) perchè viene posto ad una sequenza di otto 1 (ovvero 255).
- **MANTISSA.** Anche per la mantissa le modifiche necessarie vengono attuate solo se SPECIAL=1 poiché dovrà essere posta a 0 in caso di risultato infinito e a 1 in caso di NaN.

Questi ultimi controlli vengono posti in parallelo ed è possibile quindi ottenere il risultato finale dell'operazione.



Test-bench

Si è deciso di adottare l'approccio black-box o funzionale in cui i casi di test sono determinati dalla specifica. Nei test a seguire si è cercato di utilizzare casi di input differenti per testare diversi moduli del dispositivo in modo completo per verificarne il corretto funzionamento. Si è deciso di testare anche i valori limite, ovvero i valori agli estremi di ogni partizione (per verificare la gestione di NaN, infiniti e numeri denormalizzati).

Casi d'uso

I Test-bench che si è voluto riportare sono volti a coprire tutti i casi che si possono ricevere in input. Nello specifico i casi che si sono identificati sono i seguenti:

1. **somma tra numeri normali**

   Nello specifico per testare il caso di operazioni tra numeri normali si è deciso di testare i seguenti casi:

   - due numeri che non danno risultati particolari:

     X → Bin 01000100101111100011011110101110 Dec 1521.739990234375

     Y → Bin 01000100101001101010010011001101 Dec 1333.1500244140625

     SUB → 0

     Output → Bin 01000101001100100110111000111110 Dec 2854.89013671875

   - due numeri molto vicini con stesso esponente:

     X → Bin 00010000100000000000000000000001 Dec $5.048\ 10^{-29}$ (parte frazionaria maggiore)

     Y → Bin 00010000100000000000000000000000 Dec $5.048\ 10^{-29}$

     SUB → 0

     Output → Bin 00010001000000000000000000000000 Dec $1.009\ 10^{-28}$

   - sottrazione di due numeri di stesso segno :

     X → Bin 11000001000000010011101101100100 Dec -8.077

     Y → Bin 11000001011001100101101000011101 Dec -14.397

     SUB → 1

     Output → Bin 01000000110010100011110101110001 Dec 6.32

   - si testa la commutatività scambiando gli operandi del primo caso :

     Y → Bin 01000100101111100011011110101110 Dec 1521.739990234375

     X → Bin 01000100101001101010010011001101 Dec 1333.1500244140625

     SUB → 0

     Output → Bin 01000101001100100110111000111110 Dec 2854.89013671875

- sottrazione di due numeri di stesso segno :

  X → Bin 00010010100000000000000000000001 Dec 8.077 10^{-38}

  Y → Bin 00010000100000000000000000000000 Dec 5.048 10^{-29}

  SUB → 1

  Output → Bin 00010010011100000000000000000010 Dec 7.573 10^{-28}

- somma di due numeri con segno negativo :

  X → Bin 11000001011001100101101000011101 Dec -14.397

  Y → Bin 11000001000000010011101101100100 Dec -8.077

  SUB → 0

  Output → Bin 11000001101100111100101011000001 Dec -22.474

2. somma tra infiniti

- sottrazione tra infiniti di segno opposto:

  X → Bin 01111111100000000000000000000000 Dec +∞

  Y → Bin 01111111100000000000000000000000 Dec +∞

  SUB → 1

  Output → Bin x11111111aaaaaaaaaaaaaaaaaaaaaa Dec NaN

- somma tra infiniti di segno opposto:

  X → Bin 11111111100000000000000000000000 Dec -∞

  Y → Bin 01111111100000000000000000000000 Dec +∞

  SUB → 1

  Output → Bin x11111111aaaaaaaaaaaaaaaaaaaaaa Dec NaN

- somma tra infiniti negativi:

  X → Bin 11111111100000000000000000000000 Dec -∞

  Y → Bin 11111111100000000000000000000000 Dec -∞

  SUB → 0

  Output → Bin 11111111100000000000000000000000 Dec -∞

3. somma tra infiniti e numeri normali

- somma tra infinito negativo e un numero negativo:

  X → Bin 11111111100000000000000000000000 Dec -∞

  Y → Bin 11001001100101101011010000111000 Dec -1234567

  SUB → 0

Output → Bin 11111111100000000000000000000000 Dec -∞

- somma tra infinito positivo e un numero negativo:

  X → Bin 01111111100000000000000000000000 Dec +∞

  Y → Bin 11001001100101101011010000111000 Dec -1234567

  SUB → 0

  Output → Bin 01111111100000000000000000000000 Dec +∞

- sottrazione tra infinito positivo e un numero negativo:

  X → Bin 01111111100000000000000000000000 Dec +∞

  Y → Bin 11001001100101101011010000111000 Dec -1234567

  SUB → 1

  Output → Bin 01111111100000000000000000000000 Dec +∞

- sottrazione tra infinito positivo e zero:

  X → Bin 01111111100000000000000000000000 Dec +∞

  Y → Bin 00000000000000000000000000000000 Dec 0

  SUB → 1

  Output → Bin 01111111100000000000000000000000 Dec +∞

- somma tra infinito negativo e zero:

  X → Bin 11111111100000000000000000000000 Dec -∞

  Y → Bin 00000000000000000000000000000000 Dec 0

  SUB → 0

  Output → Bin 11111111100000000000000000000000 Dec -∞

4. somma tra NaN

- somma tra due NaN:

  X → Bin 01111111100101101011000111101000 Dec NaN

  Y → Bin 01111111100101101011111111101000 Dec NaN

  SUB → 0

  Output → Bin x11111111aaaaaaaaaaaaaaaaaaaaaa Dec NaN

- sottrazione tra due NaN:

  X → Bin 01111111100101101011000111101000 Dec NaN

  Y → Bin 01111111100101101011111111101000 Dec NaN

  SUB → 1

Output → Bin x11111111aaaaaaaaaaaaaaaaaaaaaa Dec NaN

5. somma tra NaN e numeri normali

- somma tra NaN e numero negativo:

  X → Bin 11111111100000000000000110000000 Dec NaN

  Y → Bin 11001001100101101011010000111000 Dec -1234567

  SUB → 0

  Output → Bin x11111111aaaaaaaaaaaaaaaaaaaaaa Dec NaN

- somma tra NaN e zero:

  X → Bin 11111111100000000000000110000000 Dec NaN

  Y → Bin 00000000000000000000000000000000 Dec 0

  SUB → 0

  Output → Bin x11111111aaaaaaaaaaaaaaaaaaaaaa Dec NaN

6. somma tra NaN e infinito

- somma tra NaN e infinito:

  X → Bin 11111111100000000000000110000000 Dec NaN

  Y → Bin Bin 01111111100000000000000000000000 Dec +∞

  SUB → 0

  Output → Bin x11111111aaaaaaaaaaaaaaaaaaaaaa Dec NaN

7. casi al limite del range di rappresentazione.

La notazione IEEE 754 consente di rappresentare solamente numeri il cui modulo è maggiore di 2^{-128} e minore di $(2^{23} - 2)2^{127}$. In questi casi quindi se il numero spera superiormente il limite di rappresentazione, il risultato deve essere infinito, se il limite viene superato inferiormente, ovvero si avrebbe $-2^{-128} < output < 2^{-128}$, l'output deve essere 0. Dato che l'input di numeri non normalizzati non è supportato dall'implementazione, i numeri normalizzati (con esponente pari a 0) possono dare output scorretti. In particolare sono stati forniti in input i seguenti segnali:

- sottrazione tra due numeri molto piccoli:

  X → Bin: 00000000101101100000001010000001 Dec: $1.67149585331 \cdot 10^{-38}$

  Y → Bin: 00000000101101100000001010000000 Dec: $1.67149571318 \cdot 10^{-38}$

  SUB → 1

  Output → Bin: x00000000000000000000000000000000 Dec: 0

- sottrazione tra due numeri uguali:

  X → Bin: 01000100101111100011011110101110 Dec: 1521.739990234375

Y → Bin: 01000100101111100011011110101110 Dec: 1521.739990234375

SUB → 1

Output → Bin: x00000000000000000000000000000000 Dec: 0

- sottrazione tra due numeri molto piccoli:

X → Bin: 00000011000000000000000001110000 Dec: $3.76163214517 \cdot 10^{-37}$

Y → Bin: 00000011000000000000000011110000 Dec: $3.76168954235 \cdot 10^{-37}$

SUB → 1

Output → Bin: x00000000000000000000000000000000 Dec: 0

- sottrazione tra due numeri molto piccoli:

X → Bin: 00000000100000000000000001110000 Dec: $1.17551004537 \cdot 10^{-38}$

Y → Bin: 00000000100000000000000011110000 Dec: $1.17552798199 \cdot 10^{-38}$

SUB → 1

Output → Bin: x00000000000000000000000000000000 Dec: 0

- somma tra numeri denormalizzati:

X → Bin: 10000000011111111111111111111100

Y → Bin: 00000000010000000000011000001101

SUB → 0

Output → Bin: x00000000000000000000000000000000 Dec: 0

- sottrazione tra un numero normalizzato e uno denormalizzato:

X → Bin: 10000000000000000000000000000011

Y → Bin: 11111111011111111111111111111111 Dec: $-\ 3.40282346639 \cdot 10^{38}$

SUB → 1

Output → Bin: x00000000000000000000000000000000 Dec: 0

- sottrazione tra numeri molto grandi:

X → Bin: 01111111011111111111111111111100 Dec: $3.40282285791 \cdot 10^{38}$

Y → Bin: 01111111000000000000011000001101 Dec: $1.70172600913 \cdot 10^{38}$

SUB → 0

Output → Bin: 01111111100000000000000000000000 Dec: +∞