Progetto di Reti Logiche

Altomari Giulio Giovanni

Moltiplicatore floating-point

IEEE 754

Corso di Ingegneria Informatica
Laurea triennale
Giugno 2025

Contenuti

Capitolo Primo

Introduzione

1.1 Il progetto

In questo progetto viene realizzato un moltiplicatore fra numeri che seguono lo standard IEEE 754, per la rappresentazione floating-point a precisione singola. Tale standard prevede la rappresentazione di numeri in virgola mobile in binario secondo la seguente formula:

$$(-1)^s \cdot 2^{e-\text{bias}} \cdot 1{,}m$$

Con s che rappresenta il segno, e l'esponente e m la mantissa; il valore di bias vale 127.

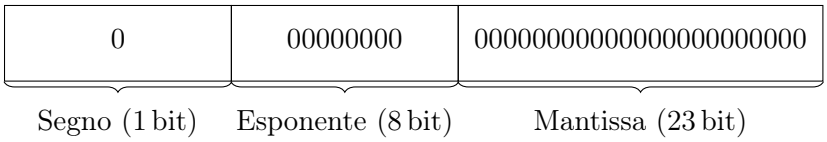

Figura 1: Codifica standard IEEE 754

1.2 Numeri Normalizzati e Denormalizzati

Lo standard IEEE 754 fa riferimento a due classi di numeri rappresentabili: Normalizzati e Denormalizzati.
La differenza è il valore rappresentabile da essi:

- I numeri normalizzati possono rappresentare i valori nel seguente intervallo:

$$x \in \left[-1.18 \times 10^{38};\ -1.18 \times 10^{-38}\right] \ \cup \ \left[1.18 \times 10^{-38};\ 1.18 \times 10^{38}\right]$$

- I numeri denormalizzati possono invece rappresentare i valori nel seguente intervallo:

$$x \in \left[-1.18 \times 10^{-38};\ -1.4 \times 10^{-45}\right] \ \cup \ \left[1.4 \times 10^{-45};\ 1.18 \times 10^{-38}\right]$$

Le due categorie sono facilmente identificabili guardando gli 8 bit appartenenti all'esponente. Nei numeri denormalizzati, tali bit sono posti interamente da 0. Altra differenza la si può trovare all'interno della mantissa, la

quale, nel caso dei numero normalizzati, si assume che abbia sempre un 1 all'inizio, che viene omesso e considerato implicito, mentre nei denormalizzati il bit implicito vale 0.

1.3 Operazioni per il calcolo della moltiplicazione

Il calcolo della moltiplicazione fra due numeri segue i seguenti passaggi riassunti:

1. **Controllo degli operandi**

   Questo passaggio controlla che gli operandi ricadano in uno dei seguenti casi:

   (a) Assumano valori invalidi come $\infty, 0, NaN$ e ricaderanno quindi nei rispettivi casi limite.

   (b) Nel caso in cui gli operandi siano in forma normalizzata o denormalizzata, procedendo con la moltiplicazione.

2. **Calcolo del segno del valore in uscita**

   Si trova il segno della moltiplicazione tra i due operandi.

3. **Moltiplicazione delle mantisse degli operandi**

   Nella moltiplicazione si include anche il bit implicito posto a 1 per i valori normalizzati e lo 0 nel caso dei denormalizzati.

4. **Somma degli esponenti e sottrazione del valore bias**

   L'esponente risultante sarà la somma dei due esponenti con la sottrazione a esso del valore di bias.

5. **Controllo caso Overflow / Underflow**

   In caso ci ritrovassimo con un Overflow o un Underflow, si codificherà il risultato con:

   (a) ∞ nel caso di Overflow.

   (b) 0 nel caso di Underflow.

6. **Scrittura del risultato**

   Il risultato verrà riportato nella medesima forma degli ingressi.

Alcune operazione di tale procedura verranno effettuate parallelamente, in quanto associate a diverse componenti dei due operandi in ingresso.

1.4 Valori Speciali

Nel caso in cui uno dei due operandi sia NaN, oppure si presentino contemporaneamente all'ingresso sia 0 che ∞, si alzerà un segnale di invalidità come previsto dallo standard.
Nel caso invece che solo uno dei due operandi sia 0 o ∞, allora il risultato seguirà la corrispondente codifica IEEE 754:

1. $+\infty$: 0 11111111 00000000000000000000000
2. $-\infty$: 1 11111111 00000000000000000000000
3. NaN : - 11111111 10000000000000000000000

Nel caso del NaN è richiesto almeno un bit della mantissa a 1, seguendo la codifica di signaling NaN l'1 è posto come MSB.
Lo zero è rappresentato intuitivamente sia con i bit di mantissa che di esponente a 0.

1.5 Gestione Casi Limite

I casi limite rappresentano il caso di Overflow e Underflow, per la loro gestione il risultato in uscita riportato sarà il limite superiore o inferiore dei valori rappresentabili.

- In caso di Overflow l'uscita sarà codificata a ∞.
- In caso di Underflow l'uscità sarà codificata a 0.

Combinazioni di ingressi che portano ad Overflow o Underflow non produrranno l'asserimento della flag di invalid.
Per la precisione il controllo dell'underflow/overflow viene fatto sull'esponente interpretato come signed e a seconda delle seguenti condizioni si compiono le relative scelte:

- Se l'esponente è maggiore di 255 avremo un Overflow, quindi il risultato sarà impostato a ∞.
- Se l'esponente è compreso tra 0 e -23 avremo un Underflow recuperabile, in quanto il risultato sarà esprimibile tramite i numeri denormalizzati.
- Se l'esponente è minore di -23 avremo un Underflow irrecuperabile, quindi il risultato sarà impostato a 0.

Capitolo Secondo

Specifica

2.1 Interfaccia del sistema

Il componente presenta l'interfaccia riportata nella figura seguente:

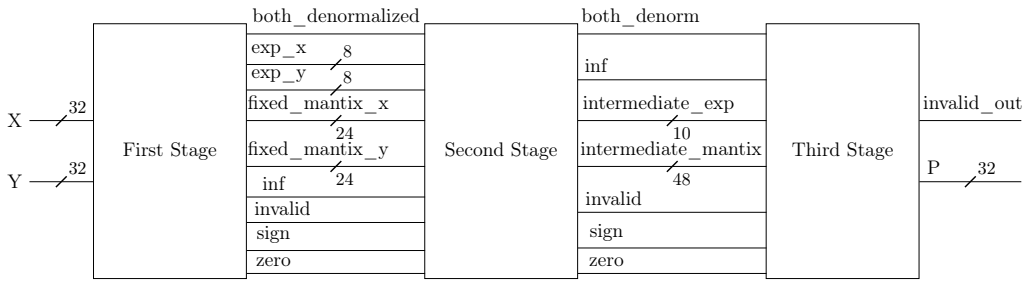

Figura 2: Interfaccia a stage del moltiplicatore

I 3 stadi eseguono le seguenti funzioni:

- **First Stage (Setup)**: Vengono divisi gli operandi dei numeri in bit di segno, esponente e mantissa, quest'ultima riceve l'aggiunta del bit implicito, in base alla tipologia degli input. Oltre a ciò vengono impostati i valori dei flag, questi riguardano il caso $zero, NaN, \infty$ e la tipologia di numero (Normalizzato o Denormalizzato).

- **Second Stage (Calc)**: Si eseguono le operazioni di moltiplica delle mantisse estese, somma degli esponenti e sottrazione di bias. In questo modo si genera un risultato parziale, il quale verrà formalizzato e corretto nello stadio seguente.

- **Third Stage (Result)**: Si valutano gli aggiustamenti da eseguire sulla mantissa e l'esponente intermedi. Qui avverrà l'arrotondamento a 23 bit e verranno esaminati i flag precedentemente impostati nel primo stadio, per controllare l'appartenenza del risultato a un caso di Overflow, Underflow o altro.
  In'uscita avremo il risultato P e un flag che segnala l'invalidità di quest'ultimo.

Una volta inseriti i due valori in ingresso sarà possibile accedere al risultato al 4° ciclo di clock.

L'architettura pipelined a 3 stadi permette di ottenere parallelismo tra il calcolo di più coppie di valori inseriti sequenzialmente, il primo risultato verrà computato dopo 3 cicli di clock, mentre ulteriori risultati derivanti da altre coppie di operandi inserite in seguito saranno resi disponibili dopo 1 ciclo di clock l'uno.

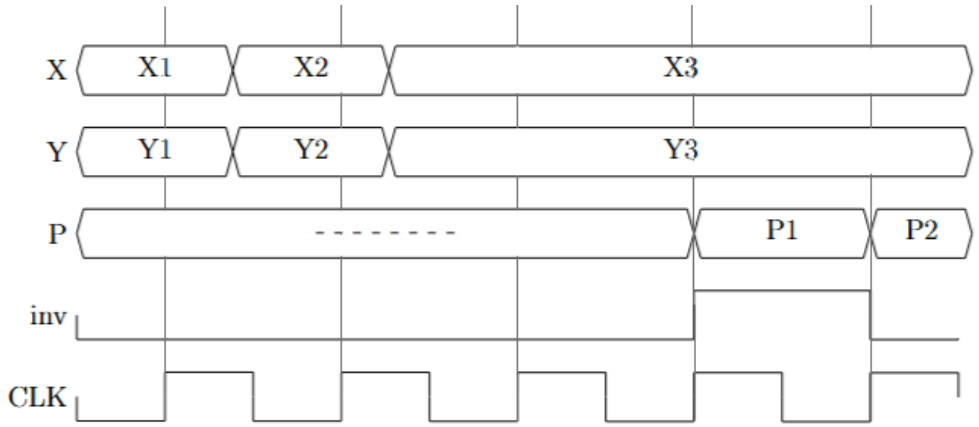

Figura 3: Diagramma temporale dell'uscita

Capitolo Terzo

Architettura del sistema

Sia i registri di ingresso e di uscita, che quelli intermedi, sono registri di tipo D, sensibili al fronte di salita del clock e dotati di reset asincrono attivo alto.

3.1 CLA

Per eseguire le operazioni di somma e sottrazione sono stati utilizzati dei moduli CLA di diverse dimensioni.

3.2 Moduli Setup Stage

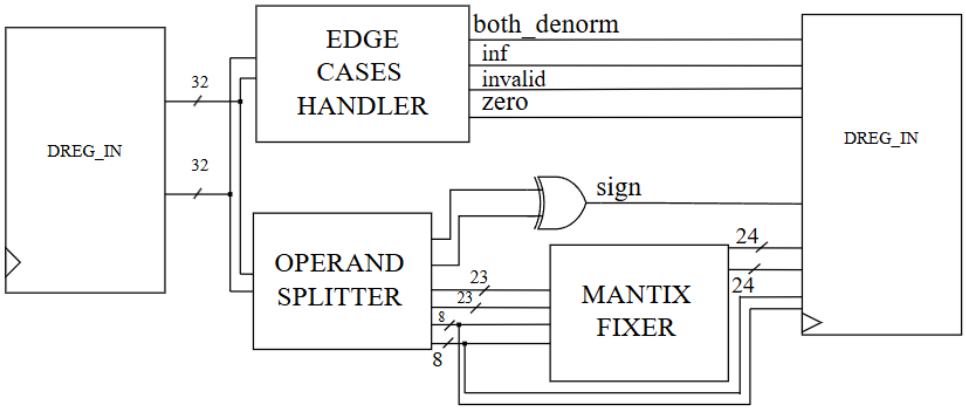

Figura 4: Architettura Setup Stage

3.2.1 Operands Splitter

Questo modulo riceve in input un numero in codifica IEEE754 e esegue la suddivisione dei campi significativi, in modo da porre in output il segno che è corrispondente al primo bit, l'esponente che sono i successivi 8 bit e la mantissa composta dagli ultimi 23 bit.

3.2.2 Edge Cases Handler

Questo modulo riceve in input i 2 numeri X e Y in codifica IEEE 754, tramite 2 moduli di Operand Splitter li suddivide nei 3 principali componenti e tramite un sub-module denominato FLAG SETTER si generano per il singolo

operando 4 segnali in uscita: zero, denorm, nan e inf. La combinazione di tutte le casistiche possibili porterà poi in output al modulo i 4 flag.

Flag Setter

I bit dell'esponente vengono posti in una porta **AND8** e **OR8**, i quali risultati vengono passati in un decoder.
All'uscita di questo decoder si trovano i flag che indicano se il numero è: **zero_or_denorm** (caso in cui l'esponente è zero, and_result = or_result = 0), **norm** (or_result = 1, and_result = 0), **inf_or_nan** (or_result = and_result = 1).
Viene eseguito un OR bit a bit della mantissa per controllare che sia presente almeno un 1, questo ci permette di ricavare un eventuale flag intermedio chiamato mantix_zero.
In ultima battuta, i flag sono ottenuti facendo rispettivamente le seguenti operazioni:

- ∞: $mantix\_zero$ **AND** inf_or_nan
- NaN: $\overline{mantix\_zero}$ **AND** inf_or_nan
- $zero$: $mantix\_zero$ **AND** $zero\_or\_denorm$
- $denorm$: $\overline{mantix\_zero}$ **AND** $zero\_or\_denorm$

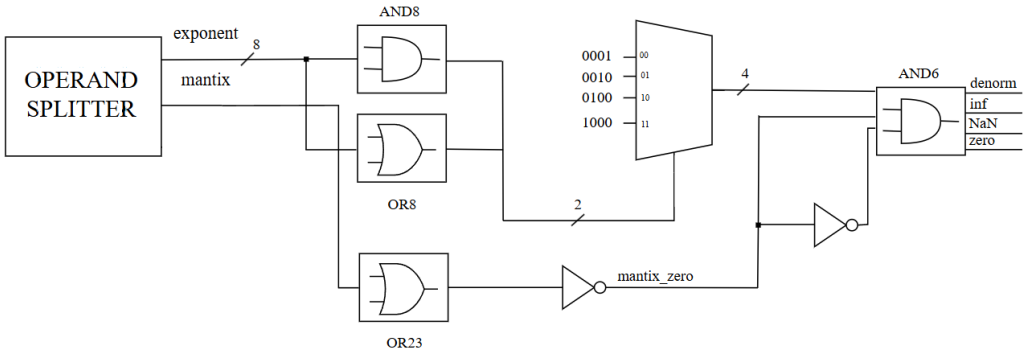

Figura 5: Flag Setter

3.2.3 Mantix Fixer

Questo componente molto semplice controlla tramite una OR8 se l'esponente ha almeno un 1, in maniera tale da distinguere se è Normalizzato o Denormalizzato, aggiungendo poi come MSB un 1 o un 0 in base al caso.

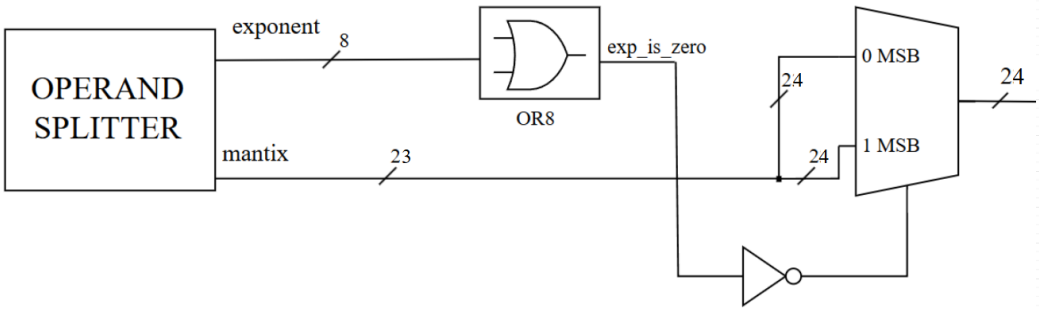

Figura 6: Mantix Fixer

3.3 Moduli Calc Stage

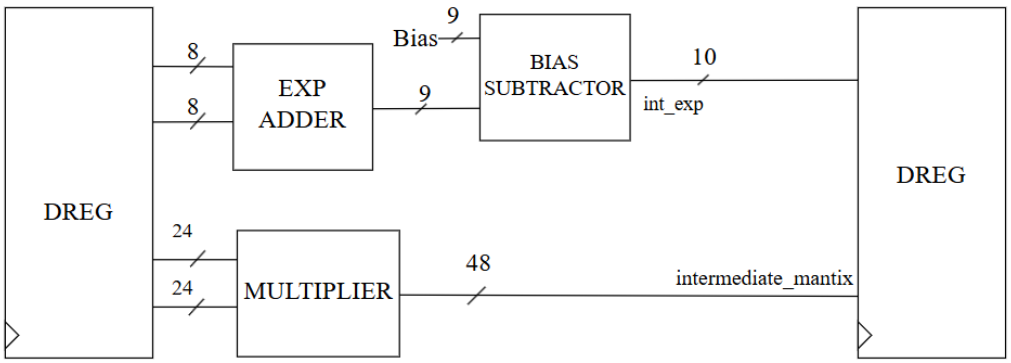

Figura 7: Architettura Calc Stage

3.3.1 Multiplier

Questo modulo si occupa della vera e propria moltiplicazione delle mantisse, prendendo in input le due mantisse da 24 bit, producendo un output da 48 bit. Il prodotto viene eseguito tramite una struttura MAC 24x24.

3.3.2 Exp Adder

Questo componente riceve in ingresso i due esponenti (8 bit) degli operandi, ed esegue una **OR8** per controllare se siano composti da soli zeri, ovvero l'operando in questione fosse un numero denormalizzato. Nel caso si abbia un numero denormalizzato in ingresso, si imposta il suo esponente a 00000001, ovvero la codifica di -126 secondo lo standard. L'uscita è ottenuta tramite un Carry Look Ahead da 8 bit, dove il riporto dell'uscita è posto come 9°

bit dell'output, questo perché è possibile che i due numeri sommati possano eccedere il valore massimo rappresentabile in 8 bit.

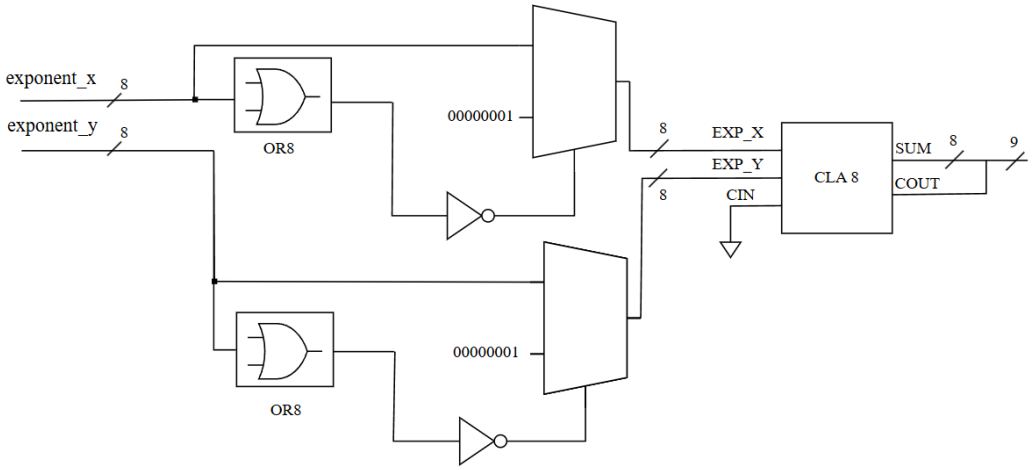

Figura 8: Exponent Adder

3.3.3 Bias Subtractor

Prende in ingresso due valori da 9 bit, la somma degli esponenti precedentemente calcolata nell'Exp Adder e il valore fisso di BIAS, ovvero 127. Il suo output è un numero in complemento a 2 da 10 bit, questo perché sarà necessario poi usare il segno per controllare un eventuale caso di Underflow. La sottrazione è effettuata tramite un CLA da 10 bit, esso è composto da un CLA a 8 bit in ripple carry con un CLA a 2 bit. Si pone all'ingresso del sommatore 127 in notazione complemento a 1 su 10 bit, l'altro ingresso riceve invece la somma dei due esponenti, con aggiunta di uno zero come MSB per uniformare le lunghezze, in quanto era unsigned. Avendo il bias in C1, si aggiunge 1 per ottenerlo in C2, passando 1 come carry-in.

3.4 Moduli Result Stage

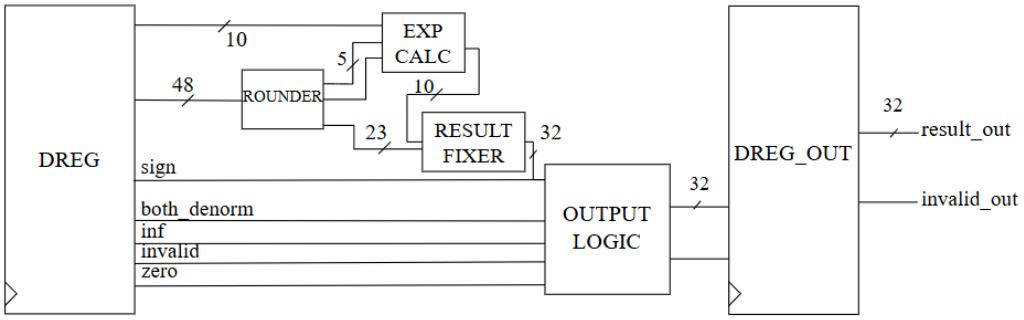

Figura 9: Architettura Result Stage

3.4.1 Rounder

Questo componente riceve in ingresso la mantissa a 48bit, risultante del prodotto fra i due operandi in ingresso al moltiplicatore. In output da il valore di offset, ovvero il numero di bit che si dovranno eliminare per ottenere la mantissa corretta;
Shifted, ovvero la mantissa correttamente traslata e un flag sub per indicare se il valore di offset andrà addizionato o sottratto all'esponente nel successivo componente Exponent Adder. Il componente si comporta nel seguente modo:

- Se incontra come 1 l'MSB allora pone il valore di offset a 1, la mantissa shifted sarà impostata sui primi 23 bit della mantissa d'ingresso e il flag sub sarà posto uguale a zero, in quanto si dovrà sommare 1 al valore d'esponente finale.

- Se incontra un 1 nel secondo bit di sinistra come primo pone offset a zero, la mantissa shifted sarà dal 47-esimo al 24-esimo bit e il flag sub zero, in questo caso l'esponente non avrà bisogno di correzione.

Tali azioni sono necessarie in quanto i due bit più significativi del risultato della moltiplicazione rappresentano valori interi (prima della virgola), per cui vanno sommati all'esponente.
Nel caso non ci trovassimo in uno dei due casi sopra citati, tramite l'implementazione dell'Offset Encoder, si prenderà come offset il risultato dato da quest'ultimo, shiftando la mantissa di offset-bit e ponendo il flag di sub a 1.
Nel caso non vi sia nemmeno un 1 nella mantissa d'ingresso dal 46-esimo al 23-esimo bit come mantissa shifted vengono riportati gli ultimi 23 bit della mantissa d'ingresso, il flag sub viene posto a 1 e offset assume il valore massimo.

Offset Encoder

Questo componente riceve una mantissa + bit esplicito, quindi 24 bit in input, in output restituisce il numero di posizioni di distanza fra il primo 1 che trova e il 24esimo bit, addizionati a 1.
Di fatto il funzionamento è quello di un Priority Encoder inverso e con un offset di 1, come per il priority encoder è presente un segnale di errore Z, utilizzato quando non viene trovato un 1 nella stringa binaria fornita.

Esempio Shifting di Mantissa:

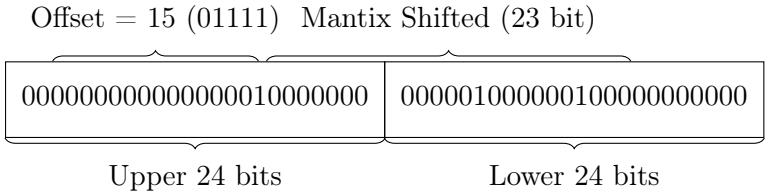

Figura 10: Shifting di mantissa

3.4.2 Final Exp Calculator

Il componente riceve in ingresso l'esponente con segno da 10 bit in uscita dal Bias subtractor, il flag Sub e l'offset dal Rounder. In uscità darà il valore dell'esponente arrotondato, ovvero il risultato tra esponente sottratto o sommato (in base al valore del flag sub) al valore di offset. L'operazione aritmetica di somma o sottrazione è svolta da un CLA10, il quale riceve in ingresso l'esponente, il valore di offset, quest'ultimo entra in una porta XOR assieme al flag di sub, in questo modo se SUB è posto a 1 pone in complemento a 1 offset. Infine in ingresso riceve anche il flag sub come Cin, in caso sia 1 questo servirà per completare l'operazione di sottrazione in complemento a 2.

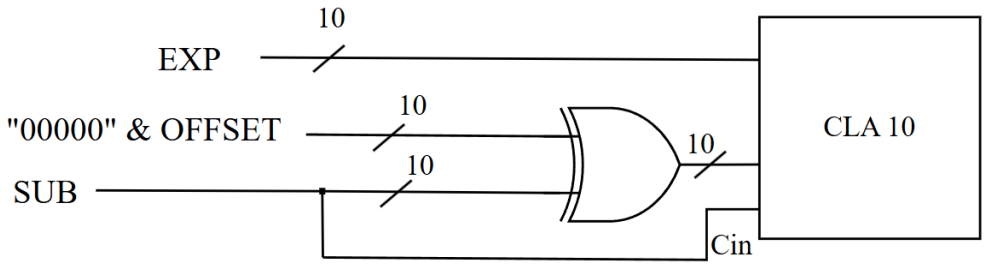

Figura 11: Final Exp Calculator

3.4.3 Result Fixer

Questo componente riceve in input l'esponente dall'Final Exp Calculator (10 bit) e la mantissa dal Rounder (23 bit). Ha il compito di controllare il valore dell'esponente con segno e in base alla casistica, modificare il valore della mantissa. I casi possibili sono i seguenti:

- Exp $>$ 254 **Overflow**: Si codifica il risultato come infinito, impostando a 1 tutti i bit dell'esponente e a 0 tutti i bit della mantissa.
- -22 $<$ Exp $<$ 0 **Denormalized Number**: Si impostano tutti i bit dell'esponente a 0, si shifta la mantissa di un valore pari alla somma di EXP e 22.
- Exp $<$ -22 **Underflow**: Si ha un underflow irrecuperabile, si imposta la mantissa e l'esponente tutto a 0.

La somma fra il valore dell'esponente e 22 si effettua tramite un CLA a 10 bit, mentre lo shift della mantissa attraverso un modulo chiamato denormalizer, il quale riceve in ingresso il valore del risultato dell'operazione sopra scritta e compie uno shift di un numero di posizioni pari al valore dato, esplicitando l'uno implicito.

3.4.4 Output logic

Questo modulo si occupa di svolgere l'ultima fase, ovvero scrivere il risultato finale. Egli riceve in input le varie flag (zero, inf, invalid_in e both_denorm) e il risultato. In base al valore dei flag ne scrive il rispettivo risultato in uscita:

- **Flag zero**: result = 0
- **Flag inf**: result = ∞

- **Flag invalid_in**: Significa che almeno uno dei due operandi in ingresso era NaN, alza il flag di invalid e il risultato è da considerarsi errato.
- **Flag both_denorm**: result = 0 (Underflow irrecuperabile)

In caso nessun flag fosse settato allora il risultato è da considerare corretto.

Capitolo Quarto

Verifica

4.1 Test-bench

Per il testing del modulo moltiplicatore floating point si è adottato un approccio "blackbox" stimolando gli ingressi con opportuni valori e controllando la correttezza del valore di uscita tramite assertions.
Gli ingressi saranno dunque dei numeri da 32 bit in standard IEEE 754 aggiornati ad ogni ciclo di clock e anche l'uscita (prodotto) segue la stessa codifica e si presenta corretta dopo 4 cicli di clock dall'ingresso relativo.
Il Test Bench del top module valuta in primo luogo tutti gli Edge Cases. dopodichè sono state scelte opportune coppie di valori di ingresso per stimolare ogni altra casistica che non produca risultati noti.

4.2 Casi d'uso

Iniziando con la valutazione degli Edge Cases è stato eseguito il testing per ogni risultato noto: (dove `invalid` non è esplicitato nel risultato è da intendersi come `0`)
Valore risultate atteso `zero` (`0 00000000 00000000000000000000000`) e `invalid` a `0`:

```
0 00000000 00000000000000000000000 * (Zero)
0 00000000 00000000000000000000000 = (Zero)
-----------------------------------
0 00000000 00000000000000000000000

0 00000000 00000000000000000000000 * (Zero)
0 11111110 11111111111111111111111 = (Norm)
-----------------------------------
0 00000000 00000000000000000000000

0 00000000 00000000000000000000000 * (Zero)
0 00000000 11111111111111111111111 = (Denorm)
-----------------------------------
0 00000000 00000000000000000000000
```

Valore risultante atteso `signed infinity` (`s 11111111 00000000000000000000000`, dove `s` è il bit di segno) e `invalid` a `0`:

```
0 11111111 00000000000000000000000 * (+inf)
0 11111111 00000000000000000000000 = (+inf)
-----------------------------------
0 11111111 00000000000000000000000

0 11111111 00000000000000000000000 * (+inf)
0 11111110 11111111111111111111111 = (Norm)
-----------------------------------
0 11111111 00000000000000000000000

0 11111111 00000000000000000000000 * (+inf)
0 00000000 11111111111111111111111 = (Denorm)
-----------------------------------
0 11111111 00000000000000000000000

1 11111111 00000000000000000000000 * (-inf)
0 11111111 00000000000000000000000 = (+inf)
-----------------------------------
1 11111111 00000000000000000000000

1 11111111 00000000000000000000000 * (-inf)
1 11111111 00000000000000000000000 = (-inf)
-----------------------------------
0 11111111 00000000000000000000000
```

Valore invalido. ci aspettiamo la flag `invalid` a `1`

```
1 11111111 10000001000010000000000 * (NaN)
1 11111111 10000001000010000000000 = (NaN)
-----------------------------------
X XXXXXXXX XXXXXXXXXXXXXXXXXXXXXXX (and invalid = 1)

1 11111111 10000001000010000000000 * (NaN)
0 11111110 11111111111111111111111 = (Norm)
-----------------------------------
```

```
X XXXXXXXX XXXXXXXXXXXXXXXXXXXXXXX (and invalid = 1)

1 11111111 10000001000010000000000 * (NaN)
0 00000000 11111111111111111111111 = (Denorm)
------------------------------------
X XXXXXXXX XXXXXXXXXXXXXXXXXXXXXXX (and invalid = 1)

1 11111111 10000001000010000000000 * (NaN)
0 00000000 00000000000000000000000 = (Zero)
------------------------------------
X XXXXXXXX XXXXXXXXXXXXXXXXXXXXXXX (and invalid = 1)

0 11111111 00000000000000000000000 * (Inf)
0 00000000 00000000000000000000000 = (Denorm)
------------------------------------
X XXXXXXXX XXXXXXXXXXXXXXXXXXXXXXX (and invalid = 1)
```

Una volta verificata l'accuratezza dei risultati tabulari il testing si è concentrato su alcune coppie di valori scelte per stimolare tutti i moduli del moltiplicatore realizzato:

Test 1: 1.0 * 1.0 = 1.0
Test banale ma di immediata verifica

```
0 01111111 00000000000000000000000 *
0 01111111 00000000000000000000000 =
------------------------------------
0 01111111 00000000000000000000000
```

Test 2: Norm * Norm = Norm
Moltiplica di due numeri Normalizzati, anche questo di facile verifica: $3.3 \cdot 7.3 = 24.09$

```
0 10000000 10100110011001100110011 *
0 10000001 11010011001100110011010 =
------------------------------------
0 10000011 10000001011100001010010
```

Test 3: Denorm * Denorm = Underflow Moltiplicare due numeri Denormalizzati darà obbligatoriamente un risultato fortemente denorma-

lizzato, sarà quindi un numero in Underflow. Per verificare il caso estremo abbiamo utilizzato il più grande numero denormalizzato rappresentabile (`0 00000000 11111111111111111111111`) = $1.18 \cdot 10^{38}$

```
0 00000000 11111111111111111111111 *
0 00000000 11111111111111111111111 =
-----------------------------------
0 00000000 00000000000000000000000 (Underflow, zero)
```

Test 4: Large Norm * Large Norm = Overflow
Moltiplicare due numeri molto grandi. i più grandi rappresentabili ($1.18 \cdot 10^{38}$). mi aspetto di ottenere Overflow

```
0 11111110 11111111111111111111111 *
0 11111110 11111111111111111111111 =
-----------------------------------
0 11111111 00000000000000000000000 (Overflow, inf)
```

Test 5: Norm * Denorm = Denorm
Questo test è improntato alla verifica del processo di "recupero denormalizzati" attuato dal rounder e dal denormalizer. il risultato intermedio di questa moltiplicazione sarà dunque un numero esprimibile come denormalizzato, i valori usati sono: $3.3 \cdot 1.498672E^{-39} = 4.945617E^{-39}$

```
0 10000000 10100110011001100110011 *
0 00000000 00100000101000110110000 =
-----------------------------------
0 00000000 01101011101101001011110
```

Test 6: Very Small Norm * Very Small Denorm, = Underflow
Questo test verifica l'altra diramazione del rounder. ovvero l'impossibilità di recupero di un numero che risulterà inferiore al minimo numero rappresentabile tramite valore denormalizzato. siamo quindi nella seconda casistica di Underflow, i valori usati sono: $7.7582626E^{-38} \cdot 1.498672E^{-39}$

```
0 00000011 10100110011001100110011 *
0 00000000 00100000101000110110000 =
-----------------------------------
0 00000000 00000000000000000000000 (Underflow, zero)
```

Test 7 Denorm * Norm
In maniera simile ai precedenti questo test mostra come sia possibile attraverso un numero Normalizzato abbastanza imponente produrre un risultato Normalizzato a fronte di una moltiplicazione con un numero denormalizzato: $9.879754E^{-39} \cdot 9.86454E^{33} = 9.7459226E^{-5}$ (Rientra nel range dei numeri normalizzati).

```
0 00000000 11010111001010010111100 *
0 11101111 11100110010111000000000 =
-----------------------------------
0 01110001 10011000110001100000110
```