Corso di laurea in Ingegneria Informatica
Politecnico di Milano

Progetto di reti logiche

Moltiplicatore floating-point IEEE754

Mattia Brianti
Alex Hathaway

Anno Accademico 2023/2024

Indice

1 Introduzione

Il presente progetto intende sviluppare un moltiplicatore per numeri in formato floating-point conformi allo standard IEEE754 in virgola mobile a precisione singola. L'obiettivo principale è sviluppare un sistema in grado di gestire operandi normalizzati, denormalizzati e valori speciali, come NaN (Not

a Number), infinito e zero, integrando al contempo un'architettura pipelined per ottimizzare l'efficienza complessiva del progetto. Il sistema dovrà eseguire come da specifica, i seguenti controlli: riconoscere che gli input siano uguali o diversi da zero, verificare l'overflow o l'underflow degli esponenti ed infine la normalizzazione e l'arrotondamento del risultato. Lo standard IEEE754 definisce la rappresentazione dei numeri in virgola mobile mediante la seguente formula:

$$(-1)^s \times 2^{e-\text{bias}} \times 1.m$$

dove s rappresenta il bit di segno, e indica l'esponente (dal quale viene sottratto il *bias*, un valore fisso pari a 127 per la precisione singola) e, infine, m rappresenta la mantissa, con un 1 implicito davanti poiché il numero è normalizzato.
Nello standard IEEE754, esistono due classi di numeri rappresentabili: i numeri normalizzati e quelli denormalizzati. Questa distinzione si basa sul range dei valori che ciascuna classe può rappresentare. I numeri normalizzati possono rappresentare i valori che vanno da $1,17 \times 10^{-38}$ a $3,4 \times 10^{38}$ e da $-3,4 \times 10^{38}$ a $-1,17 \times 10^{-38}$. Mentre i numeri denormalizzati andranno da $1,4 \times 10^{-45}$ a $1,17 \times 10^{-38}$ e da $-1,17 \times 10^{-38}$ a $-1,4 \times 10^{-45}$. La distinzione tra queste due classi può essere effettuata osservando gli 8 bit dell'esponente: nei numeri denormalizzati, infatti, l'esponente sarà costituito da tutti zeri.
Il processo di moltiplicazione tra due numeri floating-point può essere suddiviso nei seguenti passaggi fondamentali:

1. **Controllo degli operandi**: in questo passaggio si verifica se gli operandi assumono valori speciali come 0, NaN o infinito. In tali casi, viene gestito ogni caso specifico; altrimenti, se gli operandi sono normalizzati o denormalizzati, si procede con il calcolo della moltiplicazione.

2. **Calcolo del segno del risultato**: qui viene determinato il segno del numero risultante dalla moltiplicazione.

3. **Moltiplicazione delle mantisse**: in questo caso si esegue la moltiplicazione delle mantisse dei due operandi, includendo l'1 implicito per i numeri normalizzati o lo 0 implicito per quelli denormalizzati.

4. **Somma degli esponenti e sottrazione del bias**: in questo passaggio viene eseguita la somma degli esponenti dei due operandi, seguita dalla sottrazione del bias (127).

5. **Normalizzazione della mantissa**: Regolazione della mantissa risultante e adeguamento dell'esponente di conseguenza.

6. **Controllo di overflow e underflow dell'esponente**: Verifica se l'esponente rientra nel range dei numeri normalizzati, denormalizzati oppure se si é in presenza di un caso speciale come overflow o underflow.

7. **Scrittura del risultato**: Composizione finale del risultato in base ai calcoli eseguiti.

Lo standard IEEE754 definisce valori speciali per 0, Infinity e NaN:

- **0**: `00000000000000000000000000000000`
- **Infinity**: `S11111111000000000000000000000000`
- **NaN**: `S11111111-.....1.........-`

Per la codifica di NaN si intende che l'esponente sia tutti 1 e ci sia almeno un 1 nella parte della mantissa.
Nel caso in cui uno dei due operandi sia NaN o si verifichi la moltiplicazione tra 0 e infinito, il sistema genererà un segnale di invalidità conforme allo standard IEEE754. Invece, quando uno dei due operandi presenta valori come 0 o infinito, il moltiplicatore imposterà il risultato ai valori corrispondenti riportati sopra, 0 se si moltiplica per 0 ed infinito se si moltiplica per infinito. Per quanto riguarda la gestione di overflow e underflow, si é deciso di restituire come risultati i valori associati agli estremi dei valori rappresentabili. In caso di overflow, verrà restituito infinito, mentre in caso di underflow, bisognerá fare una distinzione dovuta al fatto che se l' esponente risultante é compreso tra 0 e -23 sará un underflow "recuperabile" in quanto si ricade nell' intervallo esprimibile dai numeri denormalizzati, mentre se dovesse essere < -23, si dovrá porre il risultato a 0.

2 Specifica

L'architettura del sistema prevede l'ingresso di due numeri a 32 bit in codifica IEEE754, un segnale di clock e un segnale di reset. Gli input numerici vengono posizionati in appositi registri di ingresso, mentre il risultato della moltiplicazione viene reso disponibile su un registro di uscita insieme ad una flag invalid che segnala gli edge cases esposti in precedenza. Grazie all'architettura pipelined, il sistema permette di elaborare sequenzialmente più richieste, riducendo il tempo di latenza per i risultati successivi. Essendo il sistema pipelined si é deciso di dividere il sistema nei seguenti stadi, tra i quali sono stati posti dei registri:

1. **Prep Stage**: Valuta gli input per eventuali flag (Zero, NaN, Infinity), divide gli operandi in segno, esponente e mantissa, ed estende la mantissa a 24 bit aggiungendo l'1 implicito per i numeri normalizzati o lo 0 per i denormalizzati.

2. **Calc Stage**: Esegue le operazioni aritmetiche necessarie, come la moltiplicazione delle mantisse estese e la somma degli esponenti, alla quale successivamente viene sottratto il bias.

3. **Round Stage**: Normalizza la mantissa a 23 bit e regola l'esponente in base allo shift fatto su di essa.

4. **Result Stage**: compone il risultato in uscita a 32 bit a meno che non siano presenti casi limite, in tal caso saranno segnalati in uscita.

Nel primo modulo troviamo tre componenti che svolgono le operazioni primarie della moltiplicazione tra floating point, questi sono: EDGE_CASES_CHECK, OPERANDS_SPLITTER e MANTIX_FIXER. I moduli seguenti, come indicato dai loro nomi, operano in questo modo:

- OPERANDS_SPLITTER: questo componente è responsabile della scomposizione dei 32 bit dei due segnali in ingresso in tre parti distinte: segno, esponente e mantissa.

- EDGE_CASES_CHECK: questo componente si occupa di controllare il valore degli ingressi, individuando condizioni particolari come infinito, NaN, zero e entrambi i numeri denormalizzati, tutto ció analizzando esponenti e mantisse degli operandi in ingresso.

- MANTIX_FIXER: Questo componente determina se i numeri in ingresso sono normalizzati, oppure denormalizzati, andando ad aggiungere un '1' al MSB della mantissa se si tratta di un numero normalizzato oppure uno '0'se si tratta di un denormalizzato.

Nel secondo modulo invece vengono svolte le operazioni aritmetiche necessarie al calcolo del risultato finale usando i seguenti moduli:

- EXP_ADDER : il quale somma gli esponenti tramite un CLA

- MANTIX_MULTIPLIER : questo componente moltiplica le mantisse alle quali é stato aggiunto il bit implicito

- BIAS_SUBTRACTOR : questo componente sottrae alla somma che é stata fatta in EXP_ADDER il bias (127)

Nel terzo modulo vengono modificati il prodotto delle mantisse e la somma degli esponenti in modo che venga corretto l'esponente risultante a seconda dell'arrotondamento della mantissa. Per farlo si é deciso di introdurre i seguenti componenti:

- ROUND : il componente ROUND trova la posizione del primo bit '1' in un vettore di 48 bit , genera una versione "shiftata" della mantissa di 23 bit , calcola l'offset dello shift e alza un segnale di sottrazionenei casi opportuni.
- FINAL_EXP_CALC : il componente esegue una somma e una sottrazione di un valore pari ad offset, successivamente in base ad un segnale di controllo viene selezionato il risultato corretto

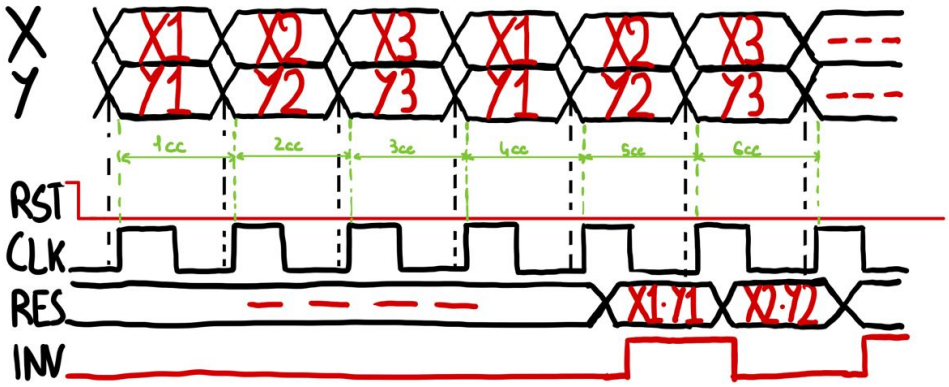
Nel quarto e ultimo modulo vengono ricombinati i risultati ottenuti in ROUND_STAGE in modo che questi vadano a formare correttamente il risultato della moltiplicazione. Per farlo si é deciso di introdurre i seguenti componenti:

- RES_FIX : il componente gestisce i casi di overflow e underflow per un numero in virgola mobile. Se c'è un overflow, il risultato è impostato ad infinito. In caso di underflow, il risultato è impostato a zero. Se il valore è denormalizzato, il componente modifica la mantissa in modo appropriato affinché compaia il risultato corretto.
- FLAG_DETECTOR : il componente gestisce valori speciali: imposta il risultato a zero per il flag di zero, a infinito per il flag di infinito, e combina segno, esponente e mantissa altrimenti, inoltre segnala valori non validi con l'output INVALID.

2.1 Interfaccia del sistema

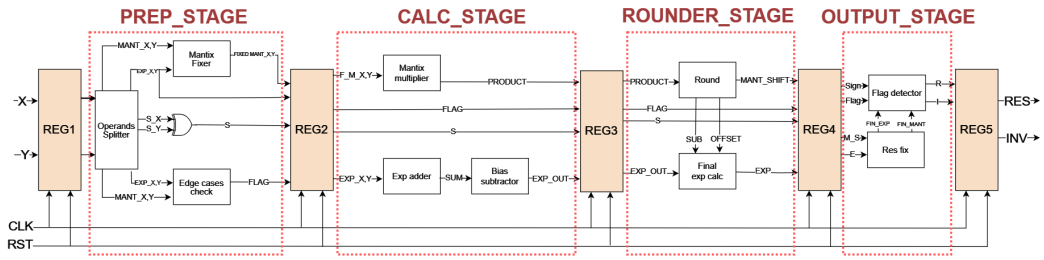
L'interfaccia progettata riceve in ingresso due numeri a 32 bit codificati secondo lo standard IEEE754, oltre a un segnale di clock e un segnale di reset, che permettaranno di implementare un'architettura pipelined. La computazione effettuata dall'interfaccia genera un risultato, anch'esso codificato in formato IEEE754, che viene reso disponibile su un registro di uscita a 32 bit, e rappresenta il prodotto dei due operandi in ingresso. In uscita, è presente anche un segnale binario (chiamato INVALID) che segnala l'invalidità del risultato: se questo segnale è impostato a valore logico alto, il valore presente nel registro di uscita non deve essere considerato corretto né utilizzabile. Il risultato della moltiplicazione sarà disponibile al quinto ciclo di clock (cc) dall'inserimento dei due valori di ingresso. L'architettura utilizzata per implementare questo processo è una pipeline suddivisa in quattro stadi,

che consente di ottenere parallelismo nel calcolo. Questo significa che, mentre il primo risultato sarà disponibile dopo 4 cicli di clock dall'inserimento degli operandi, i risultati delle coppie successive saranno disponibili con una latenza di 1 ciclo di clock ciascuno. Come rappresentato nel disegno seguente:



2.2 Architettura del MULTIPLIER_IEEE754

Il componente finale che eseguirà la moltiplicazione sarà costituito dai quattro stage precedentemente descritti, intervallati da registri intermedi, i quali condividono un segnale di RESET e un segnale di CLOCK, la cui frequenza è stata determinata dopo aver simulato la moltiplicazione con 20 diversi valori che andassero a stimolare tutte le casistiche possibili. A seguito di questa analisi, è stato stabilito un periodo di clock pari a 65 ns.
Il top module, che rappresenta l'insieme dei quattri stage, é descritto dal seguente schema a blocchi:



2.3 PREP_STAGE

2.3.1 Operands splitter

Questo componente riceve in ingresso 2 numeri X e Y in codifica IEEE754 e restituisce in uscita il segno (1bit), l'esponente (8bit) e la mantissa (23bit) di ognuno. Per la precisione il modulo utilizza come segno il bit presente nella 32esima posizione, come esponente i bit dalla 31esima alla 24esima posizione e infine come mantissa i bit dalla 23esima all'ultima posizione.

2.3.2 Edge cases check

Questo componente riceve in input l'esponente e la mantissa dei due numeri X e Y dall'Operands splitter. Il modulo tramite una porta AND8 e una porta OR8 si occupa di controllare che il valore di mantissa e esponente sia 0 o 255, in tal caso si é deciso di gestire gli operandi speciali codificandoli in questa modo:

- **Nessuna flag ("00")**: si è deciso di implementare una codifica "NoFlag" affinchè fosse possibile effettuare la moltiplicazione nei casi non identificati dalle successive Flag;
- **Infinito ("01")** : la codifica di infinito si verifica quando uno dei 2 numeri è infinito (a meno che l'altro numero non sia zero, caso trattato successivamente). Il modulo verifica se uno dei 2 numeri (o entrambi) presentino tutti i bit dell'esponente a 1 e tutti i bit della mantissa a 0 (codifica di infinito);
- **Invalid ("10")**: la codifica di invalid si verifica quando in ingresso abbiamo un numero infinito moltiplicato per uno 0, oppure quando uno dei 2 numeri è un NaN (quindi con esponente avente tutti i bit a 1 e con una mantissa con almeno un bit diverso da 0);
- **Zero ("11")**: la codifica di Zero si verifica quando uno dei 2 numeri è uno zero (escluso il caso di moltiplicazione $0 \times \infty$, già trattato precedentemente) oppure quando abbiamo 2 numeri in ingresso entrambi denormalizzati, ovvero con esponente uguale a 0.

2.3.3 Mantix fixer

Questo componente riceve in ingresso i 23bit di mantissa e gli 8 bit dell'esponente di ogni numero per poterli preparare alla moltiplicazione esplicitando il valore unitario implicito, '0' per i numeri denormalizzati e '1' per i numeri normalizzati. Questa operazione viene fatta attraverso una porta OR8 che verifica

se è presente almeno un 1 nell'esponente, trattandosi quindi di un numero normalizzato, o in caso contrario di un numero denormalizzato. Questo modulo in uscita avrà quindi le due mantisse corrette, costituite da 24bit, dove il bit aggiuntivo viene posto come MSB.

2.4 CALC_STAGE

2.4.1 Exponent adder

Questo componente riceve in ingresso gli esponenti da 8bit degli operandi e ne restituisce in uscita la somma su 9bit, poichè i due numeri sommati potrebbero avere un bit aggiuntivo di riporto. Nel caso in cui in ingresso ci fosse un numero denormalizzato, il componente imposta l'esponente di quel numero a "00000001". La somma viene effettuata utilizzando un Carry Look Ahead da 8 bit con C_{in} fissato a 0 e con il riporto d'uscita che diventa il MSB della somma degli esponenti.

2.4.2 Mantix multiplier

Il componente esegue la moltiplicazione delle mantisse dei due operandi iterando sui bit del secondo termine. Per ogni bit a '1', si genera un prodotto parziale che corrisponde al primo operando, mentre per un bit a '0', il prodotto parziale sará pari ad una stringa di 24 zeri. Per facilitare la somma dei prodotti parziali, questi vengono estesi a 47 bit mediante l'aggiunta di zeri sia a sinistra che a destra del risultato ottenuto in precedenza. Per la precisione, ogni prodotto parziale viene shiftato a sinistra in base alla posizione del bit corrispondente nel secondo operando e successivamente vengono concatenati alla sinistra del risultato un numero di zeri tali da raggiungere la lunghezza di 47 bit. Questa estensione assicura che i prodotti parziali siano della stessa dimensione e allineati correttamente per la somma, che viene eseguita utilizzando 8 componenti CSA47, che sommano tre prodotti parziali da 47 bit. Successivamente, 3 CSA48 che sommano gli 8 risultati parziali ottenuti in precedenza , con un terzo operando fissato a zero nel caso finale. L'ultimo CSA48 somma le tre somme parziali finali.

2.4.3 Bias subtractor

Questo componente prende in ingresso la somma degli esponenti da 9bit, considerata come un numero unsigned e restituisce in uscita un numero in complemento a 2 da 10bit signed, poichè sarà necessario effettuare un ulteriore controllo in caso di numeri denormalizzati o per un eventuale underflow. Il componente, in particolare, ha l'obiettivo di rimuovere il bias in eccesso,

essendo partiti da due esponenti aventi entrambi un bias iniziale. Questa sottrazione verrá effettuata con un Carry look ahead a 10bit, dove in ingresso si avrá la somma degli esponenti (con uno '0' come MSB poichè unsigned) e "1110000001" (-127 in complemento a 2). Questo ci restituirà il valore dell'esponente con segno, senza l'aggiunta del bias.

2.5 ROUNDER_STAGE

2.5.1 Round

Il componente riceve in ingresso una mantissa da 48 bit, risultato della moltiplicazione tra le due mantisse, e fornisce in uscita una mantissa shiftata, un valore 'OFFSET' e un flag 'SUB' che saranno poi utilizzati in seguito in FINAL_EXP_CALC. Le operazioni svolte dal componente sono le seguenti:

- **Caso MSB pari a 1:** se il bit più significativo della mantissa da 48 bit è uguale a 1, il valore di OFFSET verrá impostato a 1. La mantissa di uscita SHIFTED assumerá il valore dei primi 23 bit della mantissa di ingresso e succesivamente SUB verrá posto a 0.
- **Caso secondo bit significativo pari a 1:** Se il secondo bit più significativo è pari a 1 e l'MSB è pari a 0, il valore di OFFSET viene impostato a 0. In questo caso, SHIFTED assume il valore dei bit che vanno dal 47esimo al 24esimo bit della mantissa di ingresso, e SUB viene posto a 0.
- **Caso senza 1 nei bit più significativi:** se nei due casi precedenti non viene riscontrato alcun 1, OFFSET viene determinato esaminando i bit compresi tra il 46esimo e il 23esimo bit della mantissa in ingresso. La flag SUB viene impostata a 1 e la mantissa verrá poi shiftata in base alla posizione del primo 1 della mantissa da 48 bit. Se non viene trovato alcun 1 nei bit compresi tra il 46esimo e il 23esimo, allora SHIFTED assume il valore degli ultimi 23 bit di MANTIX, OFFSET viene posto a 31 (valore massimo), e SUB viene impostato a 1.

2.5.2 Final_exp_calc

Questo componente prende in ingresso l'esponente con segno, in uscita da BIAS_SUBTRACTOR, ed offset e sub in uscita da ROUND e restituisce in output il valore dell'esponente corretto, ovvero l'esponente con sommato o sottratto (in base alla flag sub) il valore di offset. Inizialmente il modulo estende l'offset ricevuto in input a 10 bit concatenando 5 zeri alla sua sinistra, in modo da poterlo sommare/sottrarre correttamente all'esponente in

ingresso. Successivamente il componente utilizza due CSA che verranno utilizzati per calcolare sia la somma che la differenza tra l'esponente in ingresso e l'offset esteso. Infine per selezionare il risultato corretto verrá utilizzato un multiplexer che in base al valore di SUB selezionerá uno dei due risultati prodotti dai CSA.

2.6 OUTPUT_STAGE

2.6.1 Res_fix

Il componente riceve in ingresso la mantissa shiftata da ROUND (23 bit) e l'esponente in uscita da FINAL_EXP_CALC (10 bit). Il suo compito é quello di corregger l'esponente e la mantissa in base alla casistica in cui ci troviamo: overflow, underflow o denormalizzato. Per capire se ci troviamo in uno degli ultimi due casi, si è utilizzato un CLA a 10 bit per sommare all'esponente il numero 22: nel caso in cui il risultato fosse negativo il numero sará da considerarsi un underflow, mentre nel caso in cui il risultato fosse positivo il numero sará da considerarsi denormalizzato. perció in sintesi questi sono i tre casi:

- **Overflow**: nel caso in cui l'esponente in ingresso fosse maggiore o uguale a 255, l'esponente verrá impostato a 255 e la mantissa a 0 (codfica di infinito)
- **Underflow**: nel caso in cui l'uscita del CLA, risulti negativa, l'esponente e la mantissa verranno fissati tutti e due a 0
- **Denormalizzato**: nel caso in cui l'uscita del CLA, risulti positiva (e non rientri nella prima casistica), l'esponente viene fissato a tutti 0, mentre la mantissa viene shiftata di un numero di bit pari al modulo dell'esponente in ingresso concatenati ad un 1.

2.6.2 Flag detector

Quest'ultimo componente riceve in input la flag (2bit) e il segno (1bit) calcolati nel primo stage, la mantissa (23bit) in uscita da ROUND e l'esponente (8bit) in uscita da FINAL_EXP_CALC e restituisce il risultato finale su 32bit e una flag di invalid, nel caso in cui il risultato fosse un valore non accettato. Il componente, in particolare, analizza la flag di ingresso:

- **Flag di Zero ("11")**: nel caso in cui l'ingresso fosse la flag di zero il componente imposterá il risultato in uscita con tutti i bit a 0;

- **Flag di Infinito ("01")**: nel caso in cui l'ingresso fosse la flag di infinito (quindi "01") il componente imposterá il risultato in uscita con il segno concatenato all'esponente avente tutti 1 e la mantissa avente tutti 0;
- **Flag di Invalid ("10")**: Nel caso in cui l'ingresso fosse la flag di invalid verrá impostata la flag di invalid su '1' e il risultato che si vedrá in uscita sará da considerarsi errato;
- **Nessuna flag ("00")**: Nel caso in cui l'ingresso fosse la flag "00", la quale non identifica casi speciali, il componente restituirá il segno concatenato all'esponente che sará a sua volta concatenato alla mantissa, andando a formare il risultato della moltiplicazione;

3 Verifica

3.1 Test-bench

Per la verifica é stato scelto di testare il componente finale stimolando gli ingressi con numeri a 32 bit e controllando che le uscite fossero uguali ai risultati aspettati. Successivamente all'ingresso degli input, bisognerá aspettare 4 cicli di clock prima di poter vedere il risultato sulla linea d'uscita.

3.2 Casi d'uso

Nel corso delle nostre attività di testing, abbiamo esaminato meticolosamente ogni scenario e caso limite. I test effettuati hanno dimostrato che il nostro moltiplicatore è in grado di gestire con successo tutte le situazioni previste.

Come punto di partenza per i test, siamo partiti dagli Edges Cases noti:

3.2.1 Risultato invalido

- $Nan \times Nan$
  $(01111111101100000000000010000000) \times (01111111101100000011000010000000)$
- $Nan \times Norm$
  $(01111111101100000000000010000000) \times (01111010101100000011000010000000)$
- $Nan \times Denorm$
  $(01111111101100000000000010000000) \times (00000000001100000011000010000000)$

- $Nan \times 0$
  $(01111111101100000000000010000000) \times (00000000000000000000000000000000)$
- $Nan \times Inf$
  $(01111111101100000000000010000000) \times (01111111100000000000000000000000)$
- $0 \times Inf$
  $(00000000000000000000000000000000) \times (01111111100000000000000000000000)$

3.2.2 Risultato "infinito con segno"

- $+Inf \times +Inf$
  $(01111111100000000000000000000000) \times (01111111100000000000000000000000)$
- $-Inf \times -Inf$
  $(11111111100000000000000000000000) \times (11111111100000000000000000000000)$
- $+Inf \times Denorm$
  $(01111111100000000000000000000000) \times (00000000011000000110000010000000)$
- $+Inf \times Norm$
  $(01111111100000000000000000000000) \times (01111010101100000011000010000000)$
- $+Inf \times -Inf = -Inf$
  $(01111111100000000000000000000000) \times (11111111100000000000000000000000)$

3.2.3 Risultato zero

- 0×0
  $(00000000000000000000000000000000) \times (00000000000000000000000000000000)$
- $0 \times Norm$
  $(00000000000000000000000000000000) \times (01111010101100000011000010000000)$
- $0 \times Denorm$
  $(00000000000000000000000000000000) \times (00000000011000000110000010000000)$

Dopo aver verificato i principali edge cases, ci siamo focalizzati sulle altre casistiche possibili per verificare al meglio il funzionamento del nostro moltiplicatore

3.2.4 $Norm \times Norm = Norm$

$X = 01000000100100000000000000000000 = 4.5$ in decimale
$Y = 01000000011100000000000000000000 = 3.75$ in decimale

$EXP = 01000001100001110000000000000000 = 16.8750$ in decimale

3.2.5 $Denorm \times Denorm = Underflow\ (0)$

$X = 00000000011111111111111111111111 = 1.1754942e - 38$ in decimale
$Y = 00000000011111111111111111111111 = 1.1754942e - 38$ in decimale

$EXP = 00000000000000000000000000000000 = 0$ in decimale

3.2.6 $Norm \times Norm = Overflow\ (infinito)$

$X = 01111111011111111111111111111111 = 3.4028235e38$ in decimale
$Y = 01111111011111111111111111111111 = 3.4028235e38$ in decimale

$EXP = 01111111100000000000000000000000 = Inf$ in decimale

3.2.7 $Norm \times Denorm = Denorm$

$X = 01000000100000000000000110000100 = 4,00018500E + 00$ in decimale
$Y = 00000000000010000010000110000100 = 7,46707E - 40$ in decimale

$EXP = 00000000001000001000011001110010 = 2,986966E - 39$ in decimale

3.2.8 $Norm \times Denorm = Underflow\ (0)$

$X = 00000001000001001000011111111111 = 1.1964442e - 38$ in decimale
$Y = 00000000000001001000011111111111 = 2.09498e - 40$ in decimale

$EXP = 00000000000000000000000000000000 = 0$ in decimale

3.2.9 $Norm \times Denorm = Norm$

$X = 01000011111000010000000000000000 = 4.500E + 02$ in decimale
$Y = 00000000000001011000110110101100 = 5.1000E - 40$ in decimale

$EXP = 00000010100111000011000010001010 = 2.295000E - 37$ in decimale