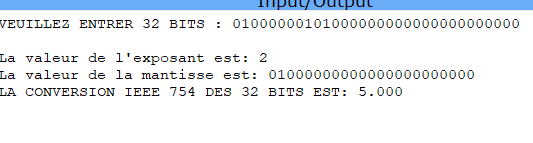
**Explication du fonctionnement du programme:**

Ce programme en assembleur pour le processeur PEP/8 convertit une séquence de 32 bits au format IEEE 754 en nombre décimal, tout en validant le format de l’entrée utilisateur. D'abord, il invite l'utilisateur à entrer une séquence de 32 bits, puis vérifie chaque caractère pour s'assurer qu'il s'agit bien de 0 ou de 1. Si un caractère invalide est détecté ou si la séquence ne contient pas exactement 32 bits, le programme affiche un message d'erreur approprié. Une fois la séquence validée, le programme procède à la conversion de l’exposant en ajustant le biais de 127 et il procède à la conversion en nombre décimal de la mantisse en suivant les spécifications du format IEEE 754. Il affiche ensuite la représentation décimale du nombre en tenant compte de la partie entière et de la partie fractionnaire.

**Documentation des tests:**

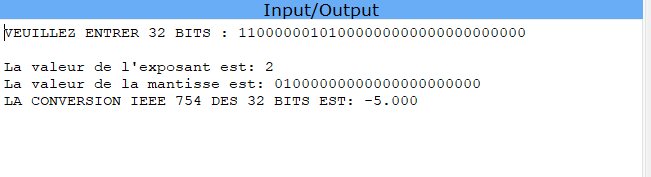
**test 1:** nombre positif



**explication**:

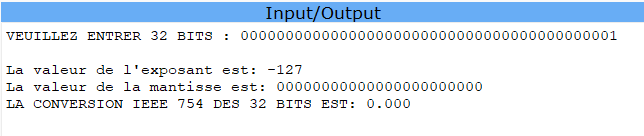
L’exposant est de 2 alors 2 bit de la mantisse seront pousser vers les entiers. Celà nous donne 101 dans les entiers et 0 dans les nombre décimaux. De plus, le bit de signe est de zéro donc la valeur est positive. Aucune erreur n’est présente.

**test 2**: nombre négatif



**explication**: L’exposant est de 2 alors 2 bit de la mantisse seront pousser vers les entiers. Celà nous donne 101 dans les entiers et 0 dans les nombre décimaux. De plus, le bit de signe est vaut un alors la valeur est négative. Aucune erreur n’est présente.

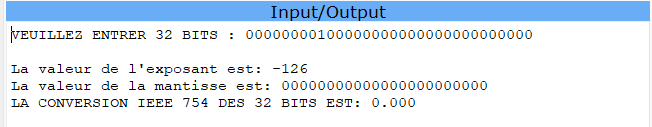
**test 3:** cas subnormal, nombre très petit



**explication**:

Comme l’exposant est exactement égal à zéro en binaire, la mantisse est dénormalisée, ce qui signifie que la valeur de la mantisse est utilisée telle quelle, avec un 0 implicite devant au lieu d’un 1 implicite. Cela permet de représenter des nombres plus petits que le plus petit nombre normal. Le code permet donc de convertir ces nombres subnormaux. Dans le test 3, le nombre converti correspond à 1.4x10^-45, donc le code affiche un nombre avec une partie fractionnaire très proche de 0.

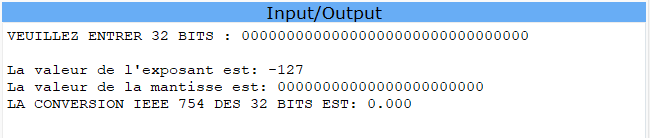
**test 4**: cas limite, plus petit nombre positif normal



**explication:**

Le nombre binaire entré correspond au plus petit nombre positif normal. Il correspond à une mantisse de 1.0 en binaire multiplier par 2 avec un exposant de -126. Le code affiche donc une partie fractionnaire qui a une valeur très proche de 0. Dû à l’arrondissement du programme, la valeur est tronqué après 3 zéros dans les décimales.

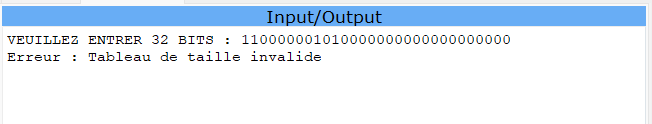
**test 5**: cas limite, zéro



**explication:**

La valeurs des 32 bits est de zéro, ce qui signifie que le nombre est positif, que l’exposant est de -127 après l’avoir converti avec le biais de 127 et que la matisse est également égale à zéro, même après la conversion avec la normalisation IEEE 754. Donc, le nombre sous forme décimale est de 0.000.

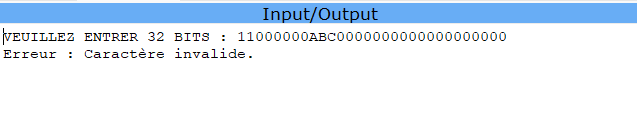
**test 6**: entrée trop courte



**explication:**

Le programme valide qu’il y a exactement 32 bits d’entrée. Si ce n’est pas le cas, il affiche un message d’erreur.

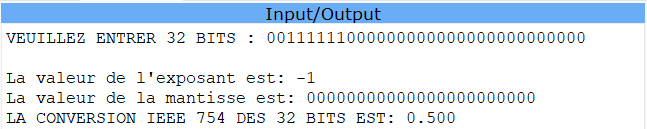
**test 7**: caractères non valides



**explication:**

Le programme valide que tous les bits d’entrée sont bien sous forme binaire, soit sous forme de 0 ou de 1. Si ce n’est pas le cas, il affiche un message d’erreur.

**test 8:** fraction



**explication :**

La valeur de la partie décimale est de zéro, mais celle de la partie fractionnaire est de 5, puisque le nombre représenté par les 32 bits en binaire correspond au nombre 0.5 en format IEEE 754.

**Choix d’implémentation:**

1. **Validation des entrées :**

La première étape s’assure que l’utilisateur entre exactement 32 bits, et que chaque bit est soit un 0 soit un 1.

Si c’est pas bon (par exemple, trop court ou avec des caractères bizarres), on affiche un message d’erreur pour que l’utilisateur comprenne où ça bloque.

1. **Conversion de l'exposant :**

L’exposant est ajusté avec un biais de 127, comme le standard IEEE 754 le demande.

On a aussi pensé aux cas particuliers, comme les nombres très petits (subnormaux) ou les nombres énormes (grands exposants).

1. **Traitement de la mantisse :**

La mantisse est convertie en décimal, avec un 1 implicite pour les nombres normaux ou un 0 implicite pour les subnormaux.

Une boucle spécifique calcule les bits de la mantisse de pour avoir un résultat exact.

1. **Calcul de la partie décimale :**

On a mis en place une méthode simple et efficace pour calculer la partie décimale en multipliant et ajustant les bits de la mantisse.

On gère aussi les cas particuliers comme les ajustements d’arrondis ou les très petits nombres.

1. **Affichage des résultats :**

Le programme montre clairement le résultat final en format décimal, tout en prenant en compte les cas comme zéro, les nombres négatifs ou les très petits nombres subnormaux.

Chaque étape d’affichage est bien organisée pour rester facile à lire et à modifier si besoin.

1. **Nomenclature propre:**

On a utilisé une nomenclature pour nommer les étiquettes et les sous-programmes. Résultat : le code est facilement lisible, et c’est facile de retrouver ce qu’on cherche ou de corriger un problème.

1. **Gestion des erreurs :**

À chaque étape critique, le programme vérifie si tout va bien. S’il y a un problème, il affiche un message clair et revient proprement à l’étape précédente.

**Contributions individuelles de l’équipe:**

**Alexandre Bouchard:** section sur la vérification (code, commentaires, nomenclature)

+ documentation

**Tristan Larue:** section sur l’exposant et la mantisse (code, commentaire, nomenclature)

**Rose-Anne Dubois:** section sur la partie décimale (code, commentaire, nomenclature)

+ documentation

**Émile Simoneau:** section sur les outputs (code, décimale, nomenclature) + documentation