École de technologie supérieure

Auteur : Iannick Gagnon

Collaborateurs : Pierre Bélisle, Francis Bourdeau

TCH017 – TP1 : Interpréteur IEEE754 simple précision Travail en équipe de 2 ou 3

Instructions

- Vous devez remettre une seule archive (**.zip**, **.rar**) par équipe qui contient tous vos fichiers.
- La **fonctionnalité**, la **lisibilité**, la **documentation** et **l'efficacité** du code seront évaluées suivant les modalités spécifiées par l'enseignant-e.

Objectifs

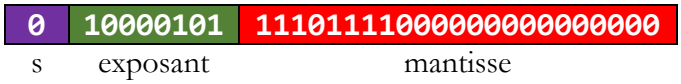
- Mettre en pratique la matière du cours afin de résoudre un problème concret.

Description théorique de la problématique

Un nombre rationnel en format IEEE754 simple précision (float) occupe 32 bits découpés de la manière suivante en partant de la gauche vers la droite :

- Le bit 31 correspond au **signe**
- Les bits 30 à 23 correspondent à l'**exposant**
- Les bits 22 à 0 correspondent à la **mantisse**

Représentation de 123.75 en float:



Rappelons la formule permettant d'interpréter une valeur IEEE 754 simple précision :

$$\mathbf{valeur} = (-1)^{s} * 1.\mathbf{Mantisse} * 2^{\mathbf{exp}-127}$$

Toutefois, en Pep/8, nous n'avons pas accès à une unité de calcul en virgule flottante permettant de calculer ces produits et puissances.

Voici comment nous procédons :

a) Signe

Lorsque s vaut 1, la valeur est négative.
Lorsque s vaut 0, la valeur est positive.

Dans cet exemple, puisque $s = 0$, nous avons un chiffre positif.

b) Exposant

La valeur de l'exposant nous renseigne sur la position du séparateur de décimales dans la mantisse. Soustrayez 127 à la valeur encodée dans l'exposant pour connaître cette position.

Exposant : $10000101_2 = 133_{10}$
Position du séparateur : $133_{10} - 127_{10} = 6_{10}$

c) Mantisse

La mantisse contient simultanément les bits de la partie entière du nombre et de la partie décimale.
La partie entière correspond à 1_2 suivi des N premiers bits de la mantisse, N étant la position calculée en b).
La partie décimale correspond aux bits restant dans la mantisse.

Dans l'exemple de la page précédente, puisque la mantisse vaut 11101111_2 et que nous avons trouvé $N = 6$. La séparation entre la partie entière et décimale peut être visualisée ainsi :

111011	11
entière	décimale

La partie entière vaut donc : 1_2 suivi de 111011_2, soit $1111011_2 = 123_{10}$.

La partie décimale vaut donc : 11_2 ou $1/2^1 + 1/2^2 = 0.75$.

Si on combine le signe positif trouvé en a), la partie entière et la partie décimale trouvée en c), nous avons recalculé la valeur 115.75 à partir de son expression binaire en IEEE754 simple précision.

En Pep/8, il est possible de stocker un float en utilisant deux mots mémoire et en les découpant en deux parties.

Reprenons l'exemple de la valeur 123.75 :

Pour stocker ce dernier en mémoire, on le divise en deux parties : (1) les bits 31 à 16 et (2) les bits 15 à 0.

0100001011110111	1000000000000000

Celles-ci correspondent à `0x42F7` et `0x8000`.

Partie 1 – Démonstration de votre compréhension du problème

Démontrez que vous serez en mesure d'interpréter la valeur d'un nombre IEEE754 simple précision sans faire de multiplication ni de puissance en vous servant de la technique décrite dans la description théorique.

Nom : _____ **gr :** _____

Nom : _____

Nom : _____

Refaites cet exercice avec la valeur 0x42BD4000 et déterminez la valeur décimale représentée dans ce float. Remettez votre démarche et la réponse à votre enseignant-e.

Quels sont les 32 bits de ce float : _____

Le bit de signe possède quelle valeur : _____

Quels sont les bits de l'exposant : _____

Quelle est la valeur de l'exposant : _____

Quelle est la position du séparateur de décimale : _____

Quels sont les bits de la mantisse : _____

Quels sont les bits et la valeur de la partie entière : _____

Quels sont les bits et la valeur de la partie décimale : _____

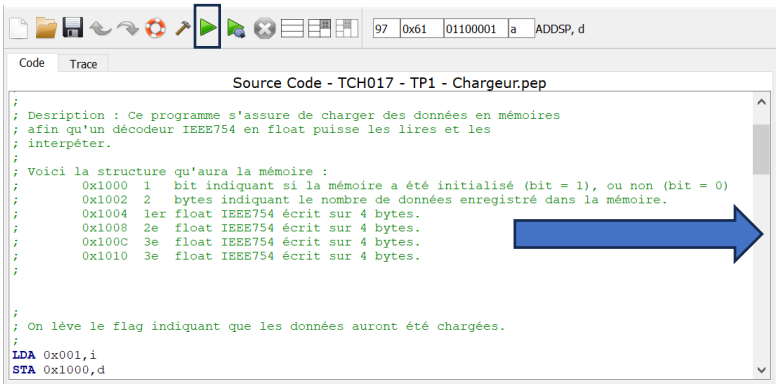
Partie 1 – Conclusion

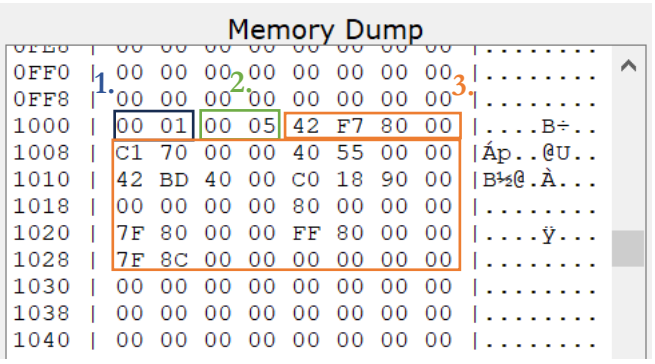
Sauvegardez une copie de ce travail en suivant le format suivant :

`equipe_xx_quiz.docx`

Description du projet

Cette section décrit comment le chargeur de données fourni dans le code de départ fonctionne.





1. À l'adresse **`0x1000`**, la valeur est inscrite lorsque les données sont chargées. Ceci indique que la mémoire a été initialisée et est prête à être utilisée. Une valeur 0 à cet endroit signale que la mémoire n'a pas été initialisée.
2. À l'adresse **`0x1002`**, on retrouve le nombre de valeurs que devra traiter votre programme.
3. À partir de l'adresse **`0x1004`**, une série de 4 octets formant plusieurs nombres écrits sous le format IEEE754.

```
; 0x1004  1er float IEEE754 écrit sur 4 bytes.
; 0x1008   2e float IEEE754 écrit sur 4 bytes.
; 0x100C   3e float IEEE754 écrit sur 4 bytes.
; 0x1010   4e float IEEE754 écrit sur 4 bytes.
; etc...
```

Votre tâche consiste à lire cette série de nombres qui auront été chargés en mémoire et d'afficher leurs valeurs à la ligne de commande (c.-à-d la fenêtre **Output** de Pep/8).

Partie 2 – Gestion de la base de données

Cette partie du travail consiste à accéder aux adresses qui contiennent les données chargées en mémoire partant des principes suivants :

1. Deux métas donnés, stockés sur 2 octets, sont contenus aux adresses 0x1000 et 0x1002
2. Puis, les nombres IEEE754 simples précision sont stockés de manière consécutive à partir de l'adresse 0x1004.
3. Chaque nombre IEEE754 simple précision occupe 4 octets.

Tâche 1.1	Valider le chargement

Description : Le mot situé à `0x1000` contient un indicateur qui signale si le chargement s'est bien déroulé. Si la valeur contenue est égale à 1, vous devez afficher le message suivant dans **Output** suivi d'un saut de ligne :

`Les données ont été chargées avec succès!`

Si la valeur est différente de 1, affichez le message suivant :

`ERREUR : Les données n'ont pas été chargées!`

Dans ce dernier cas, vous devez immédiatement arrêter le programme.

Validation : Vous devez observer le message de succès lorsque vous exécutez le programme tel quel. Pour tester le code d'erreur, remplacez **temporairement** la valeur suivante par la valeur en rouge dans le chargeur et relancez l'exécution pour observer le message d'erreur :

```
;
; Stocker l'indicateur que les données ont été chargée
;
LDA ~~0x01~~ 0x00,i
STA 0x1000,d
```

Une fois que cela est fait, réécrivez la valeur d'origine égale à `0x01`.

Tâche 1.2	Déterminer le nombre de valeurs stockées

Description : Le mot situé à `0x1002` contient le nombre de données qui ont été stockées par le chargeur. Vous devez lire cette valeur et la stocker dans une variable. Ensuite, affichez un message de la forme suivante en vous assurant que la valeur numérique (p. ex. 5) correspond à la valeur lue dans la mémoire et pas une constante littérale :

`Nombre de valeurs chargées : 5`

Validation : Lorsque vous exécutez votre programme, vous devez observer les sorties suivantes dans **Output** :

```
Les données ont été chargées avec succès!
Nombre de valeurs chargées : 5
```

N'oubliez pas le saut de ligne après la deuxième ligne.

Tâche 1.3	Calculer les adresses de départ des chiffres de la base de données

Description : Le premier chiffre commence à `0x1004` suivi du deuxième à `0x1008` et ainsi de suite. Ajoutez une boucle qui calcule chacune de ces adresses en faisant le bond approprié à chaque itération et qui les affiche une par ligne.

Validation : Lorsque vous exécutez votre programme, vous devriez observer les sorties suivantes dans **Output** :

```
Les données ont été chargées avec succès
Nombre de valeurs chargées :
Chiffre 1 @ 4100
Chiffre 2 @ 4104
Chiffre 3 @ 4108
Chiffre 4 @ 4112
Chiffre 5 @ 4116
```

Remarque 1 : Les nombres affichés par Pep/8 avec l'instruction `DECO` sont affichés en décimales, alors que nous les avons présentés en hexadécimale depuis le début de ce travail.

L'adresse 4100_{10} est la même adresse que 0x1004.
L'adresse 4104_{10} est la même adresse que 0x1008.
L'adresse 4108_{10} est la même adresse que 0x100C.
L'adresse 4112_{10} est la même adresse que 0x1010
L'adresse 4116_{10} est la même adresse que 0x1014.

Remarque 2 : Encore une fois, il est important que ces valeurs soient **calculées** et qu'il ne s'agisse pas de constantes littérales. Par exemple, si nous ajoutons une valeur au chargeur, votre code doit la détecter automatiquement.

Tâche 1.4	Extraire les deux mots de chaque chiffre

Description : Le premier mot du premier chiffre commence `0x1004` et le deuxième à `0x1004 + 0x0002 = 0x1006`. À chaque itération, lisez ces deux mots, stockez-les dans des variables nommées **`partie1`** et **`partie2`** et affichez-les de manière à reproduire l'affichage de la section **Validation**.

Par exemple, la première valeur du chargeur correspond à `0x42F78000`. Le premier mot correspond donc à `0x42F7` et le deuxième à `0x8000`. Ainsi, la première partie vaut 17143 et la deuxième vaut -32768.

Validation : Vous devez observer les sorties suivantes dans **Output** :

```
Les données ont été chargées avec succès
Nombre de valeurs chargées : 5

Chiffre 1 @ 4100
    Partie 1 : 17143
    Partie 2 : -32768

Chiffre 2 @ 4104
    Partie 1 : -16016
    Partie 2 : 0

Chiffre 3 @ 4108
    Partie 1 : 16469
    Partie 2 : 0

Chiffre 4 @ 4112
    Partie 1 : 17948
    Partie 2 : 15616

Chiffre 5 @ 4116
    Partie 1 : -16360
    Partie 2 : -28672
```

Remarque : Les nombres lus en hexadécimale dans la mémoire sont affichés en format décimal dans la console.

Partie 2 – Conclusion

Une fois que tout fonctionne correctement. Sauvegardez une copie de votre code en suivant le format suivant :

`equipe_xx_partie_1.pep`

Créez-en une copie nommée **`equipe_xx_partie_2.pep`** dans laquelle vous ferez la partie suivante.

Partie 3 – Extraction du signe et de l'exposant

Dans cette partie, vous serez amené-e-s à extraire le signe et l'exposant du nombre IEEE754 simple précision. Ces éléments correspondent aux 9 bits les plus à gauche de la partie1 du nombre.

Tâche 2.1	Extraire le signe

Description : Le bit le plus à gauche du premier mot correspond au signe. Vous devez l'extraire et le stocker dans une variable nommée **`signe`**. Vous devez aussi l'afficher comme il est montré dans la section **Validation**.

Validation : Vous devez observer les sorties suivantes dans **Output** avec les nouvelles parties en rouge dans l'énoncé seulement :

```
[…]

Chiffre 1 @ 4100
    Partie 1 : 17143
    Partie 2 : -32768
    Signe    : 0

Chiffre 2 @ 4104
    Partie 1 : -16016
    Partie 2 : 0
    Signe    : 1

Chiffre 3 @ 4108
    Partie 1 : 16469
    Partie 2 : 0
    Signe    : 0

Chiffre 4 @ 4112
    Partie 1 : 17948
    Partie 2 : 15616
    Signe    : 0

Chiffre 5 @ 4116
    Partie 1 : -16360
    Partie 2 : -28672
    Signe    : 1
```

Tâche 2.2	Extraire l'exposant

Description : Les bits 14 à 7 de la première partie d'un nombre IEEE754 simple précision contiennent l'exposant qui doit être extrait et stocké en **décimale** dans une variable nommée **`expos`**.

Deux stratégies s'offrent à vous :

Option 1 : Un masque binaire peut être appliqué afin d'isoler les bits associés à l'exposant dans la partie1, puis vous devez les décaler vers la droite pour les réaligner avec les bits de poids faible.

Reprenons l'exemple de la section théorique :

	s	exposant	Début de la mantisse
Partie 1	0	10000101	1110111
Masque	0	11111111	0000000

r	0	10000101	0000000

$r_{décalé}$	00000000 10000101

La variable $r_{décalé}$ possède une valeur de 133, ce qui est la valeur attendue.

Option 2 : Vous pouvez consulter les bits de la Partie 1, de la gauche vers la droite, en retirant continuellement le bit de gauche en multipliant par 2 et en consultant le registre de condition **`C`**. Le premier terme additionné est donc un multiple de $2^7 = 128$, le deuxième est un multiple de $2^6 = 64$ et ainsi de suite.

Validation : Consultez la page suivante →

Validation : Vous devez observer les sorties suivantes dans **Output** avec les nouvelles parties en rouge dans l'énoncé seulement :

```
Les données ont été chargées avec succès
Nombre de valeurs chargées : 5

Chiffre 1 @ 4100
    Partie 1 : 17143
    Partie 2 : -32768
    Signe    : 0
    Exposant : 133

Chiffre 2 @ 4104
    Partie 1 : -16016
    Partie 2 : 0
    Signe    : 1
    Exposant : 130

Chiffre 3 @ 4108
    Partie 1 : 16469
    Partie 2 : 0
    Signe    : 0
    Exposant : 128

Chiffre 4 @ 4112
    Partie 1 : 17948
    Partie 2 : 15616
    Signe    : 0
    Exposant : 140

Chiffre 5 @ 4116
    Partie 1 : -16360
    Partie 2 : -28672
    Signe    : 1
    Exposant : 128
```

Tâche 2.3	Calculer la puissance

Description : Vous devez calculer la puissance en soustrayant le biais à l'exposant que vous venez de calculer. Stockez sa valeur dans une variable nommée **`puiss`**.

Validation : Vous devez observer les sorties suivantes dans **Output** avec les nouvelles parties en rouge dans l'énoncé seulement :

```
Les données ont été chargées avec succès
Nombre de valeurs chargées : 5

Chiffre 1 @ 4100
    Partie 1  : 17142
    Partie 2  : -32768
    Signe     : 0
    Exposant  : 133
    Puissance : 6

Chiffre 2 @ 4104
    Partie 1  : -16016
    Partie 2  : 0
    Signe     : 3
    Exposant  : 130
    Puissance : 3

Chiffre 3 @ 4108
    Partie 1  : 16469
    Partie 2  : 0
    Signe     : 0
    Exposant  : 128
    Puissance : 1

Chiffre 4 @ 4112
    Partie 1  : 17948
    Partie 2  : 15616
    Signe     : 0
    Exposant  : 140
    Puissance : 13

Chiffre 5 @ 4116
    Partie 1  : -16360
    Partie 2  : -28672
    Signe     : 1
    Exposant  : 128
    Puissance : 1
```

Partie 3 – Conclusion

Une fois que tout fonctionne correctement. Sauvegardez une copie de votre code en suivant le format suivant :

`equipe_xx_partie_3.pep`

Créez-en une copie nommée **`equipe_xx_partie_4.pep`** dans laquelle vous ferez la partie suivante.

Partie 4 – Extraction de la partie entière

Dans cette partie, vous serez amené-e-s à calculer la partie entière du chiffre lu. Par exemple, prenez la quatrième valeur de la base de données : `0x461C3D00`. Le premier mot correspond à `0x461C` ou 17948 en base 10 et `0x3D00` correspond à 15616. Le premier mot correspond à $0100\ 0110\ 0001\ 1100_2$ et donc le bit de signe vaut 0 et l'exposant vaut $1000\ 1100_2$ ou 140. La puissance correspond à $140 - 127 = 13$. Ces valeurs correspondent au quatrième chiffre affiché dans **Output** :

```
Chiffre 4 @ 4100
    Partie 1  : 17948
    Partie 2  : 15616
    Signe     : 0
    Exposant  : 140
    Puissance : 13
```

Dans le schéma suivant, les bits retirés sont marqués en rouge et les 13 bits qui forment l'entier sont marqués en bleu :

0x461C	0x3D00
0100 0110 0001 1100	0011 1101 0000 0000

N'oubliez pas que la mantisse du format IEEE754 prend la forme `1.Mantisse` et qu'il faut ajouter un bit activé à gauche de celle-ci : $0010\ 0111\ 0000\ 1111_2$. Deux zéros furent ajoutés à gauche pour compléter l'octet. Cette valeur correspond à 9999.

Attention! Remarquez que la partie entière se trouve dans les deux mots du binaire. Cela découle du fait que les 9 premiers bits contiennent le signe et l'exposant et qu'il ne reste que $16 - 9 = 7$ bits sur 13 pour l'entier dans cette partie. Il faudra dont récupérer les $13 - 7 = 6$ premiers bits de la deuxième partie pour compléter l'entier.

Nous pouvons voir directement que la partie décimale sera égale à $0 \cdot 1/2^1 + 1 \cdot 1/2^2 = 0.25$ pour un total de 9999.25. Le calcul de la partie décimale sous la forme d'une fraction est le sujet de la prochaine section.

Tâche 3.1	Calculer la partie entière

Description : Vous devez calculer la partie entière en sachant que la puissance (`puiss`) correspond au nombre de bits à lire dans la mantisse**. Référez-vous à l'introduction de cette section pour les détails du calcul.**

Validation : Vous devez observer les sorties suivantes dans **Output** avec les nouvelles parties en rouge dans l'énoncé seulement :

```
[…]

Chiffre 1 @ 4100
    Partie 1       : 17142
    Partie 2       : -32768
    Signe          : 0
    Exposant       : 133
    Puissance      : 6
    Partie entière : 123

Chiffre 2 @ 4104
    Partie 1       : -16016
    Partie 2       : 0
    Signe          : 3
    Exposant       : 130
    Puissance      : 3
    Partie entière : 15

Chiffre 3 @ 4108
    Partie 1       : 16469
    Partie 2       : 0
    Signe          : 0
    Exposant       : 128
    Puissance      : 1
    Partie entière : 3

Chiffre 4 @ 4112
    Partie 1       : 17948
    Partie 2       : 15616
    Signe          : 0
    Exposant       : 140
    Puissance      : 13
    Partie entière : 9999

Chiffre 5 @ 4116
    Partie 1       : -16360
    Partie 2       : -28672
    Signe          : 1
    Exposant       : 128
    Puissance      : 1
    Partie entière : 2
```

Partie 4 – Conclusion

Une fois que tout fonctionne correctement. Sauvegardez une copie de votre code en suivant le format suivant :

`equipe_xx_partie_4.pep`

Créez-en une copie nommée **`equipe_xx_partie_5.pep`** dans laquelle vous ferez la partie suivante.

Partie 5 – Extraction de la partie décimale

Dans cette partie, vous serez amené-e-s à calculer la partie décimale du chiffre lu. Nous prenons le premier chiffre de la base de données comme exemple, soit 123.75 ou 0x42F78000. Dans le schéma suivant, les bits retirés pour le signe, l'exposant et la partie entière sont marqués en rouge et les 10 bits qui forment la partie décimale sont marqués en bleu :

0x42F7	0x8000
0100 0100 1111 0111	1000 0000 0000 0000

Les bits en vert seront ignorés et nous utiliserons toujours 10 bits pour la partie décimale.

On remarque que, comme pour l'exemple de partie entière, il y a des bits dans les deux parties. Le nombre de bits de la partie décimale qui se trouvent dans la première partie se calcule en soustrayant le nombre de bits retiré à la taille d'un mot : $16 - 1 - 8 - 6 = 1$. **Vous devez donc utiliser 1 bit dans le premier mot et les 9 autres dans le deuxième. Le 6 correspond à la puissance calculée de la manière usuelle avec l'exposant : $133 - 127 = 6$.**

La stratégie utilisée pour calculer la partie décimale consiste à représenter cette dernière comme la somme suivante :

$$\frac{\text{numérateur}}{\text{dénominateur}} = b_{-1} \times \frac{1}{2^1} + b_{-2} \times \frac{1}{2^2} + \cdots + b_{-10} \times \frac{1}{2^{10}}$$

Le numérateur débute à 0 et le dénominateur débute à 1. Quand b_i est actif (c.-à-d. égal à 1), on multiplie le numérateur par 2 et on lui ajoute une unité : $\text{numérateur} = 2 \cdot \text{numérateur} + 1$. On multiplie ensuite le dénominateur par 2. Quand b_i est inactif, on multiplie le numérateur et le dénominateur par 2 également.
Par exemple, supposons une partie décimale représentée par les bits 10110 (noté b_{-1} à b_{-5}), le calcul correspond à :

$$1 \times \frac{1}{2^1} + 0 \times \frac{1}{2^2} + 1 \times \frac{1}{2^3} + 1 \times \frac{1}{2^4} + 0 \times \frac{1}{2^5}$$

Puisque $b_{-1} = 1$, $\text{numérateur} = 2 \cdot 0 + 1 = 1$ et $\text{dénominateur} = 1 \cdot 2 = 2$, donc :

$$\frac{\text{numérateur}}{\text{dénominateur}} = \frac{1}{2}$$

Puisque $b_{-2} = 0$, on double le numérateur et le dénominateur :

$$\frac{\text{numérateur}}{\text{dénominateur}} = \frac{1 \cdot 2}{2 \cdot 2} = \frac{2}{4}$$

Puisque $b_{-3} = 1$, $\text{numérateur} = 2 \cdot 0 + 1 = 1$ et $\text{dénominateur} = 1 \cdot 2 = 2$, donc :

$$\frac{\text{numérateur}}{\text{dénominateur}} = \frac{2 \cdot 2 + 1}{4 \cdot 2} = \frac{5}{8}$$

Puisque $b_{-4} = 1$, $\text{numérateur} = 2 \cdot 0 + 1 = 1$ et $\text{dénominateur} = 1 \cdot 2 = 2$, donc :

$$\frac{\text{numérateur}}{\text{dénominateur}} = \frac{5 \cdot 2 + 1}{8 \cdot 2} = \frac{11}{16}$$

Puisque $b_{-5} = 0$, on double le numérateur et le dénominateur :

$$\frac{\text{numérateur}}{\text{dénominateur}} = \frac{11 \cdot 2}{16 \cdot 2} = \frac{22}{32} \approx 0.6875$$

Tâche 4.1	Calculer la partie décimale

Description : Vous devez calculer le numérateur et le dénominateur de la fraction qui correspond à la partie décimale en utilisant la stratégie présentée dans l'introduction de cette section.

Validation : Vous devez observer les sorties suivantes dans **Output** avec les nouvelles parties en rouge dans l'énoncé seulement :

```
[…]

Chiffre 1 @ 4100
    Partie 1       : 17143
    Partie 2       : -32768
    Signe          : 0
    Exposant       : 133
    Puissance      : 6
    Partie entière : 123
    Numérateur     : 768
    Dénominateur   : 1024

Chiffre 2 @ 4104
    Partie 1       : -16016
    Partie 2       : 0
    Signe          : 1
    Exposant       : 130
    Puissance      : 3
    Partie entière : 15
    Numérateur     : 0
    Dénominateur   : 1024

Chiffre 3 @ 4108
    Partie 1       : 16469
    Partie 2       : 0
    Signe          : 0
    Exposant       : 128
    Puissance      : 1
    Partie entière : 3
    Numérateur     : 336
    Dénominateur   : 1024

Chiffre 4 @ 4112
    Partie 1       : 17948
    Partie 2       : 15616
    Signe          : 0
    Exposant       : 140
    Puissance      : 13
    Partie entière : 9999
    Numérateur     : 256
    Dénominateur   : 1024

Chiffre 5 @ 4116
    Partie 1       : -16360
    Partie 2       : -28672
    Signe          : 1
    Exposant       : 128
    Puissance      : 1
    Partie entière : 2
    Numérateur     : 393
    Dénominateur   : 1024
```

Partie 5 – Conclusion

Une fois que tout fonctionne correctement. Sauvegardez une copie de votre code en suivant le format suivant :

`equipe_xx_partie_5.pep`

Combinez vos fichiers (c.-à-d. les 4 parties) dans une archive (**.zip**, **.rar**) nommée **`equipe_xx_tp1`** et déposez-le dans le dépôt prévu à cet effet ou tel que spécifié par l'enseignant-e.