3 Exercices sur les flottants

Dans toute cette section, on considérera des nombres flottants, donnés par leur représentation par signe, mantisse, exposant. On rappelle ici le nombre de bits utilisés dans les formats classiques (simple précision 32 bits et double précision 64 bits), et on donne également une représentation de flottants sur 9 bits « maison » que l'on utilisera pour certains exercices (ça fait moins de chiffres !) Dans la représentation ci-dessous, l'exposant décalé est un entier naturel, et on utilise un décalage correspondant à $2^{\text{taille de l'exposant décalé}-1} - 1$.

format	signe	exposant décalé	décalage	mantisse	signification (nombre normalisé)
32 bits	1 bit	8 bits	$2^{8-1} - 1 = 127$	23 bits	$(-1)^{\text{signe}} \times 1, \underbrace{\cdots}_{\text{mantisse}} \times 2^{\text{exposant décalé}-127}$
64 bits	1 bit	11 bits	$2^{11-1} - 1 = 1023$	52 bits	$(-1)^{\text{signe}} \times 1, \underbrace{\cdots}_{\text{mantisse}} \times 2^{\text{exposant décalé}-1023}$
9 bits	1 bit	4 bits	$2^{4-1} - 1 = 7$	4 bits	$(-1)^{\text{signe}} \times 1, \underbrace{\cdots}_{\text{mantisse}} \times 2^{\text{exposant décalé}-7}$

On rappelle qu'un *nombre normalisé* a son exposant décalé qui n'est ni $0\cdots0$, ni $1\cdots1$.

Exercice 15. *Quelques représentations.* On considère la représentation sur 9 bits donnée plus haut. À quoi sont égaux les nombres suivants ?

1. 110010000
2. 000111010
3. 011101111
4. 101111010

Exercice 16. *Représentations de dyadiques.* Donnez la représentation des dyadiques suivants sur 9 bits. On garantit qu'on peut les représenter de manière exacte.

1. 16.0
2. 0.3125
3. -8.5

Exercice 17. *Approximation.* Donner la représentation sur 9 bits du flottant le plus proche de π. (On pourra s'aider d'une calculatrice...)

Exercice 18. *Nombres représentables normalisés.* On considère une représentation avec 1 bit de signe, e bits d'exposant et m bits de mantisse.

1. Combien de nombres normalisés peut-on représenter ?
2. Quel est le plus grand nombre que l'on peut représenter ? Le plus petit ?
3. Quel est le plus petit nombre représentable strictement supérieur à 1 ? Le plus petit strictement positif normalisé ?

On rappelle maintenant ce que sont les nombres *dénormalisés.* Ceux-ci ont leur exposant décalé égal à $0\cdots0$. Si la mantisse est nulle, le nombre vaut zéro (il y a alors un zéro positif et un zéro négatif), sinon l'interprétation est

$$(-1)^{\text{signe}} \times 0, \underbrace{\cdots}_{\text{mantisse}} \times 2^{-2^{\text{taille de l'exposant décalé}-1}+2}$$

En d'autres termes, l'interprétation est la même que pour les normalisés (car l'exposant décalé vaut 0), mais le décalage est réduit de 1.

Exercice 19. *Quelques nombres dénormalisés.* On considère les nombre dénormalisés suivants, sur 9 bits. À quoi sont ils égaux ?

1. 000001111
2. 100000101
3. 000000001

Exercice 20. *Nombres dénormalisés représentables.* On considère une représentation avec 1 bit de signe, e bits d'exposant et m bits de mantisse.

1. Combien de nombres dénormalisés peut-on représenter ?
2. Quel est le plus grand nombre dénormalisé que l'on peut représenter ? Quelle est sa différence avec le plus petit normalisé positif ? Justifier la valeur différente du décalage.
3. Quel est le plus petit nombre dénormalisé strictement positif ?

Exercice 21. *Comparaison de flottants.* Montrer que pour comparer deux flottants positifs, il suffit de les comparer bit à bit. (on excluera le cas où l'un au moins des exposants décalés est $1\cdots1$)

Exercice 22. *Nombres non représentables.* Les nombres non décimaux ne sont pas représentables exactement avec des flottants. Proposer des nombres entiers ou décimaux non entiers qui ne le sont pas non plus dans les cas suivants :

— parce qu'ils sont trop grands ou trop petits.

— pour des raisons de précision.

Lorsque l'exposant décalé d'un flottant est égal à $11\cdots1$*, on parle de NAN (Not A Number). Les NAN sont utilisés pour signaler des opérations non valides (par exemple le calcul de* $\sqrt{-1}$*). Une exception : si la mantisse est nulle, le flottant représente* $+\infty$ *ou* $-\infty$ *suivant son signe. En C par exemple, le calcul de* $1/0$ *produit* $+\infty$ *(on obtient une erreur en Python). Les infinis sont utilisés pour représenter des résultats de calculs trop grands en valeur absolue.*