

Représentation binaire des nombres

juillet 2020

Remerciements à Pierre Corbineau pour son aide.

Ouvrez le fichier `nombres.py` fourni et exécutez le code :

- La fonction `to_bits(fmt, X)` permet de transformer la valeur `X` en une chaîne de 0 et de 1 qui correspond à sa représentation en mémoire selon le format `fmt`.
- Réciproquement, `from_bits(fmt, X)` reçoit dans `X` une chaîne de 0 et de 1 et la décode en supposant qu'elle représente un nombre selon le format choisi.

La notion de *format* est bien sûr cruciale ici, car elle conditionne la façon dont sont interprétées les suites de bits. Par exemple,

```
>>> to_bits('>b',65)
'01000001'
```

permet d'obtenir la représentation de 65 en base 2 (sur 8 bits). Bien sûr, si on effectue la conversion réciproque, on retrouve 65 :

```
>>> from_bits('>b', '01000001')
65
```

L'objet de ce TP n'est pas de cataloguer ni de savoir manipuler *tous* les formats binaires : on se contentera ici de représentation des entiers dans un premier temps, puis des flottants dans un second temps, sur un nombre de bits fixé.

Le caractère `>` signifie qu'on place les bits de poids fort à gauche, comme dans l'écriture usuelle en base 10.

1 Nombres entiers sur 8 bits

EXERCICE 1 (ADDITION BINAIRE)

1. Écrire une fonction `addb` qui reçoit deux entiers sous la forme de chaînes de 0 et de 1 (de longueur 8 toutes les deux), et renvoie leur somme sous cette même forme.
On suppose pour l'instant que les entiers reçus sont positifs, et que leur somme ne dépasse pas le format utilisé.
2. À l'aide de `to_bits` et `from_bits`, écrire une fonction de test « automatique » pour `addb` : on doit pouvoir donner deux entiers en base 10, effectuer l'addition en binaire d'une part et directement en `int` d'autre part, et vérifier que le résultat est conforme.
3. Traiter maintenant le cas des entiers négatifs : que faut-il changer ? On rappelle qu'en cas de débordement, on se contente d'« oublier » la dernière retenue (le bit de poids fort).
4. Écrire une fonction permettant de calculer l'opposé d'un entier écrit en binaire.
En déduire une méthode de soustraction.

2 Flottants

Pour les flottants, on utilisera le format `'>d'` :
`>>> print(to_bits('>d', 1.0))`

EXERCICE 2

1. Sur combien d'octets sont représentés les flottants avec le format `'>d'` (qui signifie double) ?
2. Retrouver dans le résultat de `to_bits('>d', 1.0)` les différents éléments qui constituent la représentation de 1.0 en binaire.
3. Quelle est l'écriture binaire de -1.0 ? Vérifier.
4. Même question pour 0.5 et pour -6.0

Tout au long de l'exercice suivant, n'oubliez pas de tester régulièrement vos fonctions, à l'aide de `to_bits` et `from_bits` si nécessaire.

EXERCICE 3 (ADDITION BINAIRE DE FLOTTANTS)

1. Écrire trois fonctions permettant de récupérer respectivement le signe, la mantisse et l'exposant d'un double, chacun sous forme binaire.
Vous pouvez réaliser ces fonctions par simple découpage de la valeur de retour de `to_bits`, ou bien travailler directement sur la valeur du nombre.
2. Afin de pouvoir additionner des flottants, il faut « réaligner » leurs mantisses en fonction de leurs exposants respectifs.
Combien de bits doit contenir une mantisse « étendue » pour tenir compte de toutes les valeurs possibles de l'exposant d'un double ? N'oubliez pas qu'il faudra également rajouter le 1 implicite.
3. Écrire une fonction qui prend en argument un double, et qui renvoie une mantisse étendue tenant compte de l'exposant, et dans laquelle l'unité a été explicitée (sans tenir compte du signe ici).
On pourra négliger le cas du zéro.
4. Écrire la fonction « réciproque » qui, à partir d'une mantisse étendue, produit un double (toujours sans tenir compte du signe).
Là encore, on peut supposer que la mantisse reçue est non nulle.
5. Écrire enfin une fonction permettant d'additionner deux double (non nuls).
Vous pouvez réutiliser la fonction écrite à la fin de la partie sur les entiers. Si vous ne l'avez pas traitée, ne tenez pas compte du signe (n'additionnez que des entiers positifs, ou à la rigueur de même signe).

Pour aller plus loin

EXERCICE 4

Adapter l'algorithme de Malcolm pour déterminer les valeurs extrêmes de l'exposant avec lequel est représenté un nombre flottant sur une machine donnée.