

# Représentation des nombres entiers

INFORMATIQUE COMMUNE - TP n° 2.7 - Olivier Reynet

## À la fin de ce chapitre, je sais :

- ☞ convertir un entier dans une base quelconque
- ☞ manipuler les bases 10, 2 et 16
- ☞ expliquer comment fonctionne un circuit qui additionne deux entiers
- ☞ utiliser des entiers signés
- ☞ gérer des dépassements de capacité

## A Manipulations

- A1. Convertir le nombre  $42_{10}$  en binaire.
- A2. Convertir le nombre  $10101010_2$  en base 10.
- A3. Convertir le nombre  $333_{10}$  en hexadécimal.
- A4. Une adresse MAC est associée à la carte réseaux de votre ordinateur. Celle-ci peut ressembler à la chaîne de caractères suivante : BE:50:73:2A:55:2F.
- (a) À votre avis, en quelle base ces nombres sont-ils codés?
  - (b) Quelle est la taille d'une adresse MAC en octets?
  - (c) Quelle est la taille d'une adresse MAC en bits?
  - (d) Convertir cette adresse en binaire.
- A5. Une image en niveau de gris est représentée par un tableau numpy dont les éléments sont des `uint8`, c'est-à-dire 8 bits unsigned integers.
- (a) De combien de niveaux de gris (de blanc à noir) dispose-t-on pour décrire l'intensité de chaque pixel?
  - (b) Vous souhaitez moyenner chaque pixel d'une image définie comme précédemment. Vous écrivez ce code qui moyenne les 8 cases voisines et le pixel :

```
import numpy as np

def moyenne(a):
    acc = np.uint8(0)
    for i in range(3):
        for j in range(3):
            acc += a[i,j]
    return acc/9

a = np.array([[50,51,52],[51,53,55],[55,57,59]], dtype=np.uint8)
print(moyenne(a))
```

Le résultat est-il juste? Pourquoi?

- (c) Proposer une version correcte de la fonction moyenne. Quel est le type de valeur retournée par cette fonction?
- (d) Écrire l'instruction qui permet de remplacer la valeur du pixel central par sa moyenne dans le tableau `a`. Vérifier le type de donnée de la case `a[1,1]`. Que s'est-il passé?
- (e) Proposer une version de la fonction moyenne sans boucles `for`!

## B Convertir dans une base quelconque

- B1. Écrire une fonction de signature `to_binary(a : int) -> str` qui renvoie la représentation binaire d'un nombre sous la forme d'une chaîne de caractères. Par exemple, `(to_binary(35))` renvoie `"100011"`.
- B2. En déduire une fonction de signature `to_base(a : int, b : int) -> str` qui renvoie la représentation d'un nombre selon la base `b` sous la forme d'une chaîne de caractères. Par exemple, `to_base(61, 3)` renvoie `"2021"` et `to_base(333, 16)` renvoie `"14D"`.

## C Half-adder et full-adder

On cherche à simuler les circuits électroniques qui réalisent l'opération d'addition sur des entiers sur  $n$  bits. On choisit de travailler sur des entiers non signés. On définit les circuits suivants :

- Le Half Adder réalise l'addition de deux bits. Ils prend deux bits  $A$  et  $B$  en entrée et possède deux sorties :  $S$  qui vaut  $A \oplus B$  et  $C$  qui vaut  $A.B$ , la retenue.
- Le Full Adder réalise l'addition de deux bits avec retenue éventuelle à l'entrée. Il prend trois bits  $A$ ,  $B$  et  $R$  en entrée et possède deux sorties :  $S$  qui vaut  $A \oplus B \oplus R$  et  $C$  qui vaut  $(A.B) \vee (R.(A \oplus B))$ , la retenue.

$\oplus$  désigne le OU Exclusif bits à bits,  $\cdot$  le ET bit à bits et  $\vee$  le OU bits à bits.

En Python, on a les opérateurs suivants :

- `&` ET bits à bits,
- `|` OU bits à bits,
- `^` OU exclusif bits à bits.

- C1. Écrire une fonction de signature `to_binary_nbits(a : int, n : int) -> list`. Il s'agit d'une variation de la fonction `to_binary` précédente, mais qui renvoie une liste d'entier plutôt qu'une chaîne de caractères. Par exemple, `to_binary_nbits(42, 8)` renvoie `[0, 0, 1, 0, 1, 0, 1, 0]`. La liste résultat possède  $n$  éléments, même si les premiers sont nuls.
- C2. Écrire une fonction de signature `nbits_to_int(u : list) -> int` qui transforme la liste résultat de la fonction `to_binary_nbits` en entier correspondant. Par exemple, `nbits_to_int([0, 0, 1, 0, 1, 0, 1, 0])` renvoie 42.
- C3. Écrire une fonction de signature `half_adder(a : int, b : int) -> (int, int)` qui implémente le circuit Half adder.
- C4. Écrire une fonction de signature `full_adder(a : int, b : int, r : int) -> (int, int)` qui implémente le circuit Full adder.

- C5. Écrire une fonction de signature `add_words(u: list, v: list) -> list` qui fait l'addition de deux listes représentant des entiers en binaire à l'aide de la fonction `full_adder` en propageant la retenue. Par exemple, `add_words([0, 0, 1, 0, 1, 0, 1, 0], [0, 0, 0, 0, 1, 0, 0, 0])` renvoie `[0, 0, 1, 1, 0, 0, 1, 0]`.

## D Connaître la valeur signée d'entiers

Dans les questions qui suivent, on n'hésitera pas à utiliser les fonctions de calcul bits à bits :

- `<<` décalage binaire à gauche. Par exemple, `1 << 3` vaut  $2^3$ .
  - `>>` décalage binaire à droite. Par exemple, `60 >> 2` vaut 15.
  - `&` calcule le et bits à bits. Par exemple, `260 & 0xFF` vaut 4.
  - `~a` calcule le complément à 1 de `a`. Par exemple, `~(0b00100011) & 0xFF` vaut 220.
- D1. Écrire une fonction de signature `is_neg_signed(a: int, n: int) -> bool` qui teste si un entier positif `a` sur `n` bits possède une valeur négative dans le cas où il est signé. Par exemple, `is_neg_signed(214, 8)` renvoie `True`. Mais `is_neg_signed(33, 8)` renvoie `False`. Au début de la fonction, on s'assurera par une assertion que  $a < 2^n$ .
- D2. Écrire une fonction de signature `signed_nbits_to_int(a: int, n: int) -> int` qui renvoie la valeur de `a` sur `n` bits, dans le cas où `a` est signé. Par exemple, `signed_nbits_to_int(214, 8)` renvoie -42 et `signed_nbits_to_int(33, 8)` renvoie 33. ã