

Activité 4 : Représentation des entiers relatifs

Pour représenter les entiers relatifs en notation binaire, on doit étendre la représentation en binaire naturel aux nombres négatifs. Une solution pourrait être de réserver le bit de poids fort (MSB) pour le signe de l'entier à représenter et d'utiliser les autres pour représenter sa valeur absolue : 0 indiquerait un entier positif et 1 un entier négatif.

Ainsi, avec des mots de 16 bits, en utilisant 1 bit pour le signe et 15 bits pour la valeur absolue, on pourrait représenter les entiers relatifs de

$$1111\ 1111\ 1111\ 1111 = -(2^{15} - 1) = -32\ 767$$

à

$$0111\ 1111\ 1111\ 1111 = 2^{15} - 1 = 32\ 767.$$

Cependant, cette méthode a plusieurs inconvénients, l'un d'eux étant qu'il y a deux zéros, l'un positif et l'autre négatif.

Ensuite, il complique les opérations arithmétiques. Par exemple pour additionner deux entiers relatifs encodés de cette manière, il faudrait faire une addition ou une soustraction selon que les entiers sont du même signe ou non. Ainsi l'addition binaire de 0101 (5_{10}) et 1101 (-5_{10}) donne :

0	1	0	1	5_{10}		
+	1	1	0	1	-5_{10}	
	0	0	1	0	2_{10}	

*Retenue non conservée
(résultat sur 4 bits)*

Le résultat de cette opération est différent de 0, ce qui complique la mise en œuvre d'opérations.

1. Ecriture des nombres signés

La solution pour résoudre ces problèmes est d'utiliser un encodage particulier nommé **binaire signé en complément à 2**.

Dans cet encodage :

- le bit de poids fort représente le signe des entiers (0 pour un entier positif et 1 pour un entier négatif),
- la représentation des nombres positifs ou nul est inchangée
- les nombres négatifs sont encodés par la méthode du complément à deux.

Si on utilise des mots de 4 bits, on peut représenter les entiers relatifs compris entre -8 et 7. Les entiers relatifs x strictement négatifs sont représentés comme l'entier naturel $x + 2^4 = x + 16$. Ainsi, les entiers naturels de 0 à 7 servent à représenter les entiers relatifs positifs ou nuls et les entiers naturels de 8 à 15 décrivent les entiers relatifs strictement négatifs.



1. Compléter la table suivante en déterminant pour chaque quartet sont équivalent décimal dans le cas d'un encodage d'un entier relatif par la méthode du complément à 2.

Nombre binaire				Valeur en binaire naturel	Valeur en binaire signé en complément à 2
MSB			LSB		
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	
0	0	1	1	3	
0	1	0	0	4	
0	1	0	1	5	
0	1	1	0	6	
0	1	1	1	7	
1	0	0	0	8	
1	0	0	1	9	
1	0	1	0	10	
1	0	1	1	11	
1	1	0	0	12	
1	1	0	1	13	
1	1	1	0	14	
1	1	1	1	15	-1

2. Lister l'intervalle des entiers relatifs codables avec un octet. Même question avec des mots de 32 bits et 64 bits.

2. Calcul du complément à 2

Le complément à deux d'un mot binaire m de k bits s'obtient en inversant les k bits et en ajoutant 1.

Le complément à deux de $44_{10} = 0101100_2$ est donc :

	0	1	0	1	1	0	0
Complémenter ↓	1	0	1	0	0	1	1
Ajouter 1 ↓	1	0	1	0	1	0	0

Le complément à 2 d'un nombre signé transforme un nombre positif en un nombre négatif et vis versa.

Le résultat du complément à 2 de 101100_2 (44_{10}) représente donc le nombre -44_{10} :

$$-44_{10} = 1010100_2$$



3. Convertir les entiers naturels 7 et -7 en des nombres binaires signés de 4 bits dans la notation en complément à 2. Même question pour une représentation en nombres binaires signés sur 8 bits.

4. Convertir en décimal les nombres binaires signés dans la notation en complément à 2 suivants :

- 11011001
- 00110110
- 10110101

3. Programmation d'un convertisseur binaire d'entiers relatifs

Objectif

L'objectif de cette partie est de développer en langage python, un programme capable de convertir un entier relatif en un nombre binaire signé dans la notation en complément à 2.

Algorithme

L'algorithme retenu pour remplir cet objectif est le suivant :

```
convertir_dec_binaire(nbreDecimal) :  
    nbreBin ← Conversion de [nbreDecimal] en binaire naturel  
    si nbreDecimal > 0 :  
        retourner nbreBin  
    sinon :  
        nbreBinComplemente ← complément de nbreBin  
        nbreBin ← nbreBinComplemente + 1  
        retourner nbreBin
```

Travail de programmation à effectuer

Une partie des fonctions présentées dans l'algorithme ci-dessus sont fournies sur Moodle dans le fichier *convertisseur.py* . Le travail à effectuer consiste à compléter ce fichier afin de rendre opérationnel le convertisseur.

Fonction 1 : Complément d'un nombre binaire

On souhaite définir une fonction `complNbreBin(nbreBin)` qui complémente un nombre binaire exprimé sous la forme d'une liste d'entiers 1 et 0.

Exemple : `complNbreBin([0,1,1,0,1,0,0,1])` renvoie la liste `[1,0,0,1,0,1,1,0]`

5. Ecrire et spécifier la fonction `complNbreBin(nbreBin)`. **Tester** le fonctionnement de cette fonction.



Fonction 2 : Ajouter 1 à un nombre binaire

On souhaite définir une fonction `ajout1(nbreBin)` qui ajoute 1 à un nombre binaire exprimé sous la forme d'une liste d'entiers 1 et 0.

Exemple : `ajout1([1,0,0,1,1])` renvoie la liste `[1,0,1,0,0]` (calcul détaillé ci-dessous).

Pour rappel l'addition de nombres binaires s'exécute selon le même mécanisme qu'une addition en décimal. Par exemple si l'on souhaite ajouter 1_2 à 10011_2 , le calcul s'effectue en effectuant des additions successives rang par rang, en partant du rang 0 et en reportant les retenues sur le rang supérieur. On obtient ainsi :

$$\begin{array}{rcccccc} \text{Retenues} \rightarrow & 0 & 0 & 1 & 1 & & \\ & 1 & 0 & 0 & 1 & 1 & \\ + & & & & & & 1 \\ \hline & 1 & 0 & 1 & 0 & 0 & \end{array}$$

La fonction `additionBinaire(b1,b2)` disponible dans le fichier *convertisseur.py* renvoie le résultat et la retenue de la somme des deux chiffres binaires `b1` et `b2`.

6. Compléter la fonction `additionBinaire` afin de respecter la spécification présentée. Les opérateurs `%` (reste de la division) et `//` (division entière) pourront avantageusement mis en œuvre dans ce programme.

7. Ecrire et **spécifier** la fonction `ajout1(nbreBin)` en réutilisant la fonction `additionBinaire(b1,b2)` précédente. **Tester** le fonctionnement de cette fonction.

Fonction 3 : Convertir un entier relatif en binaire

On cherche maintenant à définir la fonction `convertir_dec_binaire(nbreDecimal)` qui convertit un entier relatif en binaire. Le résultat est exprimé sous la forme d'une liste d'entiers 1 et 0.

Exemple : `convertir_dec_binaire(-55)` renvoie `[1, 0, 0, 1, 0, 0, 1]`

8. A partir des fonctions définies dans le fichier *convertisseur.py*, **écrire** et **spécifier** la fonction `convertir_dec_binaire(nbreDecimal)`. **Tester** le fonctionnement de cette fonction.

