TD_complement_a_2

March 23, 2021

1 Exercice 1 : Conversion

Remarque : Vous pouvez vous auto-corriger grâce à l'outil calculatrice sur PC en mode programmeur

Convertir les entiers suivants **sur 8 bits** en complément à 2

- 57
- $1010\ 1001_{(C2)}$
- -100
- $1010\ 0001_{(C2)}$
- -123
- $0011\ 1101_{(C2)}$
- 45
- $0111\ 0010_{(C2)}$

2 Exercice 2 : Propriétés remarquables du complément à 2

Remarque : Vous pouvez vous auto-corriger grâce à l'outil calculatrice sur PC en mode programmeur (sous Windows)

1. **Convertir** en décimal :
   - $1111_{(C2)}$
   - $1111\ 1111_{(C2)}$
   - $11\ 1111\ 1111_{(C2)}$

   En déduire comment s'écrit -1 en complément à 2 sur 2 octets ? puis sur n bits ?
2. **Convertir** :
   - 12 en compléments à 2 sur 8 bits, puis sur 10 bits
   - -12 en compléments à 2 sur 8 bits, puis sur 10 bits
   - Soit un entier x écrit sur n bits en compléments à 2. A l'aide des calculs précédents **expliquer** comment écrire ce même entier x sur un nombre plus grand de bits

3 Exercice 3 : Le complément à 2 en machine

Remarque : Vous pouvez vous auto-corriger grâce à l'outil calculatrice sur PC en mode programmeur

1. **Sans faire de conversion en décimal, calculer** les opposés des entiers suivants écrits sur 8 bits en compléments à 2 :
   - $0110\ 0110_{(C2)}$
   - $1000\ 1110_{(C2)}$
   - $1001\ 1001_{(C2)}$
   - $0000\ 0010_{(C2)}$
2. **Ecrire en binaire** le nombre maximal et le nombre minimal que l'on peut écrire sur 1 octet en compléments à 2. **Calculer** leur conversion en décimal, puis **exprimer** le résultat en faisant apparaitre des puissances de 2.

4 Exercice 4 : Pourquoi l'encodage en complément à 2^n ?

4.1 Partie A

Dans cette partie, on étudie un encodage à priori plus simple et on montre pourquoi cette solution d'encodage n'est pas retenue.

L'encodage proposé consiste simplement à *réserver* le bit le plus à gauche comme **bit de signe**. Le bit de signe est égal à :

- 0 si le nombre est positif
- 1 si le nombre est négatif

Questions :

1. **Ecrire sur 4 bits** les nombres 4 et -2 en utilisant l'encodage *bit de signe*
2. **Poser l'addition binaire** entre 4 et -2. Quel est le résultat obtenu une fois converti en décimal ?
3. Quelles sont les 2 représentations possibles de l'entier nul dans cet encodage. (Astuce : on remarquera que -0 = 0)
4. **Déduire** des questions précédentes les raisons pour lesquelles cet encodage n'est pas une solution retenue en machine.

4.2 Partie B

Dans cette partie, on étudie l'origine du complément à 2^n. Pour cela, on recherche l'encodage qui respecte l'addition binaire classique.

$$\begin{array}{r} 0\ 1\ 0\ 0 \\ +\ \ldots\ \ldots\ \ldots\ \ldots \\ \hline = 0\ 0\ 0\ 0 \end{array}$$

1. **Compléter** l'addition binaire **4 bits** suivante.

   On remarque que le nombre qui complète l'addition correspond bien à l'encodage *complément à 2*.

   Sur 4 bits, l'addition ci-dessus s'écrit (*très étrangement*) en décimal $4 + 12 = 0$. Ainsi, pour respecter l'addition, "l'opposé de 4 est 12". Or $12 = 2^4 - 4 = 2^n - 4$ (avec n = 4). D'où le nom de cet encodage

   En conclusion, représenter -x sur n bits revient à représenter $2^n - x$ sur n bits "en binaire classique"

2. Avec x = 52 et n = 8 bits dans l'équation précédente, calculer "l'opposé" de 52. **Convertir** le résultat obtenu en "binaire classique" et **montrer** qu'on retrouve bien le codage en complément à 2.
3. **Ecrire** en quelques lignes les raisons matérielles pour lesquelles l'encodage en complément à 2 pour les entiers relatifs s'est imposé.

   Conclusion : Si une machine sait effectuer l'addition et le calcul de l'opposé, alors il est facile de comprendre qu'elle peut exécuter les 4 opérations de base sur les entiers.

4. **Donner** l'explication de cette conclusion, c'est-à-dire expliquer comment les soustractions, multiplications et divisions sont faciles à réaliser à partir d'additions et de calculs d'opposé.