Complément à deux et nombres signés

Le **complément à deux** est une représentation binaire qui permet d'effectuer les opérations arithmétiques usuelles naturellement.

Explications

La notation est utilisée sur des écritures de nombres de longueur donnée (nombres écrits couramment sur 8, 16, 32 ou 64 bits). Dans une telle écriture on utilise le bit de poids fort (bit le plus à gauche) du nombre pour contenir la représentation de son signe (positif ou négatif, le zéro étant considéré comme positif).

La première idée est de marquer le signe du nombre de façon simple : le signe puis la représentation de sa valeur absolue. Ainsi :

⇒ 00000010 = +2 en décimal
⇒ 10000010 = (-2) en décimal

Malheureusement cette représentation possède deux inconvénients.

- Le premier est que le nombre zéro (0) possède deux représentations: 00000000 et 10000000, qui sont respectivement égales à 0 et -0.
- L'autre inconvénient est que cette représentation n'est pas compatible avec l'addition; l'addition usuelle d'un nombre négatif et d'un nombre positif ne fonctionne pas.
- Ainsi : 00000011 + 10000100 = 10000111
  Soit 3 + (-4) = (-7) au lieu de (-1)

C'est pour remédier à ces problèmes que l'on utilise la notation en **complément à deux**.

Les nombres positifs sont représentés comme attendu; par contre les nombres négatifs sont obtenus de la manière suivante :

- On inverse tous les bits de l'écriture binaire de sa valeur absolue (opération binaire NOT), on fait ce qu'on appelle le **complément à un**,
- On ajoute 1 au résultat (les dépassements sont ignorés).

Cette opération correspond au calcul de 2^n-|X|, où *n* est la longueur de la représentation. Ainsi -1 s'écrit comme 256-1=255=11111111_2, pour les nombres sur 8 bits.

Les deux inconvénients précédents disparaissent alors. En effet, l'opération précédente effectuée sur 00000000 permet d'obtenir 00000000 (-0=+0=0) et l'addition usuelle des nombres binaires fonctionne.

La même opération effectuée sur un nombre négatif donne le nombre positif de départ: 2^n-(2^n-x)=x.
Pour coder (-4) :

- On prend le nombre positif 4 : 00000100
- On inverse les bits : 11111011
- On ajoute 1 : 11111100

Le bit de signe est automatiquement mis à 1 par l'opération d'inversion. On peut vérifier que cette fois l'opération 3 + (-4) se fait sans erreur :

00000011 + 11111100 = 11111111

Le complément à deux de 11111111 est 00000001 soit 1 en décimal, donc 11111111 = (-1) en décimal.

Le résultat de l'addition usuelle de nombres représentés en complément à deux est le codage en complément à deux du résultat de l'addition des nombres. Ainsi les calculs peuvent-ils s'enchaîner naturellement.

Exercices :

1. Compléter le tableau ci-dessous concernant un mot binaire signé de 8 bis :

Bit de signe								
								127
								2
								1
								0
								–1
								–2
								–127
								–128

2. Quelles sont les 2 valeurs extrêmes que peut prendre en décimal un mot de 8 bits non signé.

≤ N ≤

3. Quelles sont les 2 valeurs extrêmes que peut prendre en décimal un mot de 8 bits signé.

≤ N ≤

4. Compléter le tableau suivant :

Valeur binaire	Equivalent décimal en nombre non signé	Equivalent décimal en nombre signé
0000 0100		
1101 0011		
0111 1111 1000 1110		
1000 0010 0000 1101		

5. Exercices sur internet : http://www.scientillula.net/MPI/fex6_conversions/fex6_conversions.html

6. Détailler l'addition suivante : 23 + (-47)

Travaux pratiques

a. Lancer le logiciel MPLAB et ouvrir le projet **NOMBRES_SIGNES**
b. Ouvrir le fichier NOMBRES_SIGNES.c
c. Quel est le format de la variable « huit_bits_signes » ?
d. Quel est le format de la variable « huit_bits_non_signe » ?
e. Passer en mode SIMULATION
f. Compiler
g. Ouvrir la fenêtre WATCH
h. Visualiser les 2 variables en décimal
i. Mettre un point d'arrêt au début du programme
j. Exécuter le programme en mode pas à pas et compléter l'évolution des 2 valeurs. Vous devez être capable de justifier ces évolutions.

huit_bits_signes	huit_bits_non_signe

bit de signe									
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**=**	**127**
0	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**=**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**=**	**1**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**=**	**0**
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**=**	**–1**
1	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**=**	**–2**
1	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**=**	**–127**
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**=**	**–128**

Exercices sur internet :
http://www.scientillula.net/MPI/fex6_conversions/fex6_conversions.html