# Architecture des ordinateurs - TD 02

## 1 Représentation d'entiers signés

1. Donnez sur 8 bits les représentations signe plus valeur absolue, complément à 1 et complément à 2 des nombres décimaux suivants :
   $36_{10}$, $-123_{10}$, $-45_{10}$

   > **Solution:** $36_{10} = 00100100_2$ dans les 3 écritures.
   > $-123_{10} = 11111011_2$ en signe+abs, $10000100_2$ en complément à 1 et $10000101_2$ en complément à 2
   > $-45_{10} = 10101101_2$ en signe+abs, $11010010_2$ en complément à 1 et $11010011_2$ en complément à 2

2. Quelle est la valeur en décimal des nombres binaires dont la représentation en complément à 2 sur 8 bits est :
   $11111111_2$, $01001000_2$, $10001111_2$
   Quelles seraient ces valeurs si ces nombres binaires représentaient un complément à 1 ? Un signe plus valeur absolue ?

   > **Solution:** $11111111_2 = -1_{10}$ si cplment 2, $0_{10}$ si cplment 1 et $-127_{10}$ si sign+valeur abs.
   > $01001000_2 = 72_{10}$ dans les 3 modes
   > $10001111_2 = -113_{10}$ si cplment 2, $-112_{10}$ si cplment 1 et $-15_{10}$ si sign+valeur abs.

3. Considérons les représentations en complément à 2 de x, y et z sur 8 bits :
   $10010101_2$, $00010010_2$, $11101011_2$
   Sans calculer la valeur de x, y et z en décimal, donnez leur représentation en complément à 2 sur 16 bits.
   Quel est le nombre minimal de bits pouvant représenter x en complément à 2 ? Même question pour y et z.

   > **Solution:** $x = 1111111110010101$, $y = 0000000000010010$, $z = 1111111111101011$
   > Pour x, 8 bits. Pour y et z, 6 bits.

4. Déterminez les valeurs maximales et minimales représentables en complément à 2 sur n bits.

   > **Solution:** $-2^{n-1} \leq x \leq 2^{n-1} - 1$

5. Comment identifierez-vous un nombre pair sous sa représentation binaire en signe + valeur absolue ? En complément à 1 ? En complément à deux ?

   > **Solution:** En signe + val abs, il finit par 0, ainsi qu'en complément à 2. En complément à 1, le bit de signe et le dernier signe doivent être identiques.

## 2 Additions / soustractions d'entiers signés

1. Transformez la soustraction de deux entiers signés en une addition dans la représentation en complément à 2 puis calculez $120_{10} - 45_{10}$.

**Solution:** $120_{10} - 45_{10} = 120_{10} + -45_{10} = 01111000_2 + 11010011_2 = 01001011_2$

2. Sans passer par leur représentation décimale, effectuer les opérations suivantes sur des nombres de 8 bits en complément à deux :
$00100100_2 + 01000000_2$
$00011001_2 + 11001110_2$
$11110110_2 + 10100110_2$
$00101101_2 + 01101111_2$
$10000001_2 + 11000000_2$

**Solution:** Pour les overflow : c'est si les deux derniers bits de retenue sont différents.
En complément à 2 : $00100100_2 + 01000000_2 = 01100100_2$
$00011001_2 + 11001110_2 = 11100111_2$
$11110110_2 + 10100110_2 = 110011100_2$
$00101101_2 + 01101111_2 = 010011100_2$ (overflow)
$10000001_2 + 11000000_2 = 101000001_2$ (overflow)

3. En complément à 2, quel est le nombre de bits minimum nécessaires pour effectuer l'opération $125_{10} + 5_{10}$ ?

**Solution:** 9 bits (sinon overflow)

4. Le complément à 2 d'un nombre positif $x = x_{n-1}2^{n-1} + \cdots + x_0 2^0$ codé sur $n$ bits peut-être défini de deux manières :
— $CA2(x) = 2^n - x$
— $CA2(x) = \overline{x} + 1$ (le complément à 1 plus un).
Montrez que les deux définitions sont équivalentes.

**Solution:**
Soit $x = x_{n-1}2^{n-1} + \cdots + x_0 2^0$, posons

$$\begin{aligned}
2^n - x =& 2^n - (x_{n-1}2^{n-1} + \cdots + x_0 2^0) \\
2^n - x =& (2^{n-1} + \cdots + 2^0 + 1) - (x_{n-1}2^{n-1} + \cdots + x_0 2^0) \\
2^n - x =& (1 - x_{n-1})2^{n-1} + \cdots + (1 - x_0)2^0 + 1 \\
2^n - x =& (\overline{x_{n-1}}2^{n-1} + \cdots + \overline{x_0}2^0 + 1 \\
2^n - x =& \overline{x} + 1
\end{aligned}$$

## 3 Un peu de programmation : cardinal d'un ensemble représenté avec un champs de bits

Soit un ensemble $E \in \mathcal{P}([|0;31|])$ par exemple $E' = \{4, 5, 8, 15\}$. Une représentation très compacte de cet ensemble est basée sur l'utilisation d'un champs de bit (*bit set* en anglais). Dans cette représentation on encode $E$ sous la forme d'un mot de 32 bits $M = (m_{31}m_{30} \ldots m_0)_2$. On utilise la convention convention suivante : $m_n = 1$ si et seulement si $n \in E$. Par exemple, pour représenter l'ensemble $E'$ on utilisera le mot $M' = 00000000000000001000000100110000_2$.

Nous souhaitons écrire une fonction `char get_size(uint32_t M);` qui calcule le cardinal de l'ensemble $E$. Cela revient à compter le nombre de bits à 1.

1. Écrire la fonction `get_size` en utilisant une boucle et l'opérateur $>>$. Dans le pire cas vous effectuerez 32 itérations.

**Solution:**

```
char get_size(uint32_t M) {
    char count = 0;
    while(M) {
    count += M & 1; // incrémenter si le dernier bit est à 1
    M = M >> 1; // décaler m à droite
    }
    return count;
}
```

2. Considérez le mot $M'$ ci-dessus. Calculez $M'' = M'\&(M'-1)$. Calculez $M''' = M''\&(M''-1)$. Que remarquez vous ?

**Solution:** $M'' = 00000000000000001000000100110000_2\&(00000000000000001000000100101111_2) = 00000000000000001000000100100000_2$
$M''' = 00000000000000001000000100000000_2$
à chaque application de la formule on mets à zéro le bit le plus à droite.

3. Utilisez le résultat de la question précédente pour réécrire la fonction `get_size` en utilisant une boucle qui effectue au maximum $card(E)$ itérations.

**Solution:**

```
char get_size_faster(uint32_t M) {
    char count = 0;
    while(M) {
    M = M & (M-1); // on annule le bit à 1 le plus à droite.
    count += 1;
    }
    return count;
}
```

4. Question ouverte : voyez vous des algorithmes plus efficaces pour calculer $card(E)$ ?

**Solution:** Si on précalcule dans un tableau le nombre de bits à 1 dans des mots de 8 bits. Une possiblité bien plus rapide consiste à découper l'entier en 4 mots de 8 bits et utiliser la table pour obtenir le résultat.

Bien entendu, on pourrait faire la même chose directement avec une table de 32 bits, mais sa taille serait alors prohibitive.