REPRESENTATION DES NOMBRES ENTIERS RELATIFSEN INFORMATIQUE

Pour ces nombres, il faut donc stocker en plus une information sur le signe du nombre. L'ordinateur doit de plus « savoir » que ce nombre correspond à un entier relatif...

1ère (*fausse bonne*) idée :

On utilise simplement un bit, le MSB (celui le plus à gauche), pour indiquer le signe selon la convention suivante :
- bit à 0 → entier positif
- bit à 1 → entier négatif

Les autres bits indiquent la valeur absolue du nombre.

Exemple : pour un nombre sur un octet (8 bits) :

- **0**1010101$_2$ = + 85
- **1**1010101$_2$ = – 85

Problème n°1 : que représentent avec cette convention les nombres binaires suivants :

- 00000000$_2$ = **+ 0$_{10}$**
- 10000000$_2$ = **– 0$_{10}$** **→ 2 représentations différentes pour la même valeur (0), on n'aime pas en informatique !!!**

Problème n°2 : dans les systèmes informatiques, pour des questions de coût et de simplicité, c'est le même circuit électronique (appelé « additionneur ») qui réalise aussi bien les additions que les soustractions ; en effet, soustraire deux nombres équivaut à *additionner l'un à l'opposé de l'autre* :

a – b = a + (– b)

Avec la convention suivante sur le bit de signe, que donnerait alors l'opération suivante :

```
  1  1  1
 (0) 1 0 1₂   ( = 5₁₀ )
+ (1) 0 1 1₂   ( = – 3₁₀ )
-----------------------
(1) 0 0 0 0₂      ( = – 0₁₀ )      →      c'est faux !!!
```

2. La solution : le complément à deux

Pour pallier aux problèmes précédents, les entiers négatifs sont codés selon la convention suivante appelée ***complément à deux*** :

- la convention sur le bit de signe est la même que ci-dessus (0 → positif / 1 → négatif)
- les nombre positifs sont codés « normalement » (avec leur MSB à 0 bien entendu...)
- les entiers négatifs sont codés ainsi :
  - on part du nombre positif, et on inverse tous ses bits (0 ↔ 1)
  - on rajoute 1 au nombre binaire obtenu (*en laissant de côté l'éventuelle dernière retenue*)

Exemple : 24$_{10}$ = (0)11000$_2$
- inversion des bits : (1)00111$_2$
- ajout de 1 : **(1)01000$_2$** = – 24$_{10}$

1. En complément à deux, quel est maintenant la notation de « – 0 » ? Que constate-t-on alors ?
   + 0$_{10}$ = (0)0$_2$ → représentation de « – 0 » : (1)1$_2$ + 1 = (0)0$_2$: identique !
   (on laisse de côté la dernière retenue)

2. Qu'obtient-on si l'on fait l'addition 24 + (– 24) ?
   (0)11000$_2$ + (1)01000$_2$ = (0)00000 → le résultat est bien égal à 0, ça marche !!

3. Calculer de même la valeur de la différence 5 – 3 :
   3$_{10}$ = (0)011$_2$ → – 3$_{10}$ = (1)101$_2$ donc : (0)101$_2$ + (1)101$_2$ = (0)010$_2$ *(on laisse de côté la dernière retenue)*
   = + 2$_{10}$ OK !

III. Applications

1. Donner les représentations en binaire sur 8 bits des nombres suivants en complément à 2 :

- 58_{10} = **(0)011 1010$_2$**
- -1_{10} = **(1)111 1111$_2$**
- -127_{10} = **(1)000 0001$_2$**

2. Que vaut en base 10 les nombres suivants codés en complément à deux :

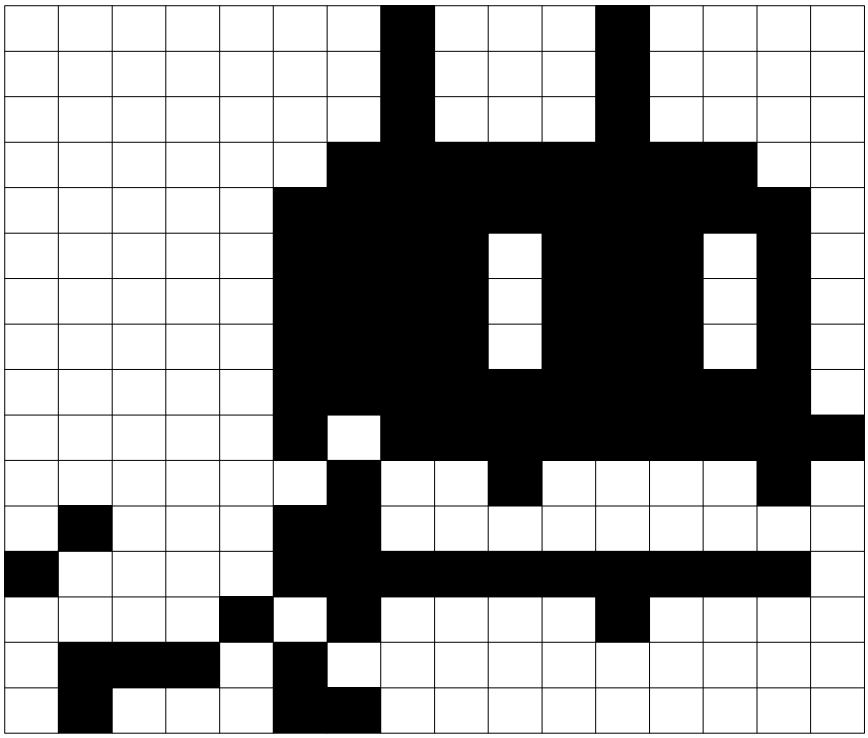
- $1001\ 1100_2 \rightarrow$ « – **(0)110 0100$_2$ »** = **– 100$_{10}$**
- $0111\ 0101_2$ = **+ 117$_{10}$**

2. Pour chacun des nombres de bits du tableau ci-dessous, indiquer :
- la valeur entière la plus grande que l'on peut coder en utilisant le complément à 2
- la valeur entière la plus petite

Nombre de bits	2	4	8	n
valeur la plus grande	**01_2 = +1_{10}**	**0111_2 = +7_{10}**	**01111111_2 = +127_{10}**	**+ $2^n/2 - 1$**
valeur la plus petite	**10_2 = – 2_{10}**	**1000_2 = – 8_{10}**	**10000000_2 = – 128_{10}**	**– 2^n / 2**

3. <u>Et pour terminer sur la représentation des entiers naturels et relatifs...</u>

L'objectif de cet exercice est de dessiner une image matricielle dans le quadrillage 16x16 ci-dessous grâce aux réponses aux questions de conversions entre les bases numériques.
Chaque case de l'image correspond à un bit. Une ligne de l'image fait 16 cases, soit 16 bits, soit 2 octets.
- Lorsque le nombre est négatif, on considérera le binaire complément à 2.
- Lorsque le bit est à 1, la case est noire, lorsque le bit est à 0, la case est blanche.



L1-L2-L3 = 272_{10} = 1 0001 0000_2
L4 = $3FC_{16}$ = 11 1111 1100_2
L5-L9 = $3D8_{16}$ + 426_{16} = $7FE_{16}$ = 11111111110_2
L6-L7-L8 = 1978_{10} = 11110111010_2
L10 = 2^{15} – $7A01_{16}$ = 8000_{16} – $7A01_{16}$ = $5FF_{16}$ = 10111111111_2
L11 = 100 1000010_2
L12-L16 = 100_{16} x 46_{16} = 4600_{16} = 100 0110 0000 0000_2
L13 = 1000 0111 1111 1110_2
L14 = 1010 0001 0000_2
L15 = 111 0010 0000 0000_2