

## TD1 : codage de l'information

### Exercice 1, complément à 2 :

- 1.1) Comment sont représentés  $(34)_{10}$  et  $(-42)_{10}$  en complément à 2 sur 8 bits? Et sur 12 bits?
- 1.2) Posez l'addition (en binaire) des deux nombres précédents.
- 1.3) Proposez une règle pour de passer de l'écriture d'un entier relatif en complément à 2 sur  $p$  bits à son écriture sur  $p + k$  bits (en conservant l'égalité des valeurs bien entendu). Justifiez votre réponse.

### Exercice 2, représentation à virgule fixe :

- 2.1) Peut on écrire exactement en binaire sur un nombre fini de bits  $1/3$ ?  $0.125$ ?  $0.1$ ?  $0.2$ ?  $0.3$ ?  $0.4$ ?  $0.5$ ?
- 2.2) Tous le nombres représentables en binaire sur un nombre fini de chiffres sont-ils représentables en décimal sur un nombre fini de chiffres?
- 2.3) Et dans l'autre sens? Justifiez les deux réponses.

### Exercice 3, représentation à virgule flottante :

On travaille avec un format flottant jouet sur 12 bits, dont le codage est le suivant :

$$c = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \hline s & & & & & & & & & & & m \\ \hline \end{array}$$

Si  $1 \leq (b)_2 \leq 14$ , le nombre représenté est  $(-1)^s \times (1, m)_2 \times 2^e$ , avec  $e = (b)_2 - 2^3$  (on ne se préoccupe pas du codage de zéro, des sous-normaux, ou des valeurs exceptionnelles ici).

- 3.1) Comment peut-on représenter  $x_1 = (11, 1)_{10}$  dans le format flottant binaire décrit ci-dessus? Vous effectuerez un arrondi au plus proche, et exprimerez votre résultat en binaire puis en hexadécimal.
- 3.2) On souhaite ajouter les deux flottants  $x_1 = (1, 1110001)_2 \times 2^2$  et  $x_2 = (1, 1100101)_2 \times 2^4$  : posez l'opération demandée de manière à obtenir le résultat avant arrondi; donnez ensuite une approximation de  $x_3$  par un flottant binaire, en effectuant un arrondi au plus proche.

### Exercice 4, détection et correction d'erreurs :

Étant donnée une source émettant une information codée sur  $n$  bits reçue par un récepteur, on définit les termes suivants :

- un codage est *auto-vérificateur* s'il permet de détecter une erreur de transmission, et
- un codage est *auto-correcteur* s'il permet de reconstituer l'information.

Trouvez des exemples simples de codages auto-vérificateurs ou auto-correcteurs (indices : preuve par 9, contrôle de parité). Quelles sont les possibilités et les limites de vos propositions?

### Exercice 5, Codes de Hamming :

Le *code auto-correcteur de Hamming* permet de corriger un bit erroné dans une information de 4 bits  $(i_3, i_2, i_1, i_0)$ . On code cette information au moyen de 3 bits supplémentaires  $p_2, p_1$  et  $p_0$  : le codage est  $(i_3, i_2, i_1, p_2, i_0, p_1, p_0)$ , où :

- $p_0$  est le bit de parité paire du sous-mot  $(i_3, i_1, i_0, p_0)$  (le bit qu'il faut ajouter pour qu'il y ait un nombre pair de 1 dans le sous-mot),
- $p_1$  est celui du sous-mot  $(i_3, i_2, i_0, p_1)$ , et
- $p_2$  est celui de  $(i_3, i_2, i_1, p_2)$ .

À la réception de  $(i_3, i_2, i_1, p_2, i_0, p_1, p_0)$ , on vérifie ces trois parités, et on définit  $t_k$  par :  $t_k = 0$  si  $p_k$  est correct et  $t_k = 1$  sinon.

Par magie,  $(t_2, t_1, t_0)$  est alors le rang dans  $(i_3, i_2, i_1, p_2, i_0, p_1, p_0)$  du bit erroné, écrit en binaire : 000 si l'information est correcte, 001 si  $p_0$  est erroné, 010 si  $p_1$  est erroné, 011 si  $i_0$  est erroné, 100 si  $p_2$  est erroné, 101 si  $i_1$  est erroné, 110 si  $i_2$  est erroné ou 111 si  $i_3$  est erroné.

On parle ici du code  $(7, 4)$  : 7 bits transmis pour 4 bits d'information effective.

- 5.1) Vérifiez le fonctionnement correct de l'algorithme de correction sur quelques exemples.
- 5.2) Justifiez la propriété auto-correctrice d'un tel codage.

- 5.3) Peut-on définir un codage auto-correcteur de 4 bits utilisant moins de 3 bits supplémentaires?
- 5.4) Tentez de généraliser l'algorithme de Hamming à d'autres tailles de mots.

**Exercice 6, lecture d'entiers au clavier :** On veut de mettre au point un morceau de programme pour la lecture d'entiers naturels au clavier, disons en C pour fixer les idées, mais vous pouvez travailler au niveau algorithmique. Le programme doit inviter l'utilisateur à entrer un entier naturel en décimal, stocker la saisie sous la forme d'une chaîne de caractères (tableau de caractères de type `char`, terminé par un 0 en C), puis calculer la valeur lue dans un entier de type `uint16_t` (type d'entiers non-signés 16 bits en C99, défini dans `stdint.h`). On note  $n$  le nombre de chiffres de l'entier lu au clavier.

- 6.1) Proposez une première version du programme qui effectue la conversion en commençant par le chiffre de poids faible. Combien de multiplications effectue la boucle de conversion de votre programme?
- 6.2) Proposez une seconde version dans laquelle la chaîne est convertie en commençant par le chiffre de poids fort. Combien de multiplications effectue la boucle de conversion?
- 6.3) Quel est le plus grand entier que l'on peut représenter dans un `uint16_t`? Que se passe-t-il si l'utilisateur de votre programme tente d'entrer un entier plus grand?

**Exercice 7, bonus :** Ecrire un programme pour compter le nombre de bits non nuls dans un entier non signé : le coût doit être proportionnel au nombre de bits non nuls (et pas au nombre de bits dans la représentation).