CORRIGÉ

1re

Représentation des données — Binaire et hexadécimal

Corrigé détaillé
4 exercices

> Ce qui est noté au contrôle, ce n'est pas le résultat seul : c'est le raisonnement posé, puis le calcul mené jusqu'au bout avec l'unité.

1 Conversions. Détailler le calcul.

1. a. On additionne les puissances de 2 des bits à 1 :
$8 + 2 + 1 = 11$
2. Soit **11**.
3. b. Le A vaut 10 :
$1 \times 16 + 10 = 26$
4. Soit **26**.
5. c. **256** valeurs, de **0 à 255** en non signé.
6. d. Chaque F vaut 15 :
$15 \times 16 + 15 = 255$
7. Soit **255**, la plus grande valeur d'un **octet**.
8. e. Parce que quatre bits représentent **16** combinaisons, exactement le nombre de chiffres de la **base 16**.
$2 \times 2 \times 2 \times 2 = 16$

2 Complément à 2 sur 8 bits. Raisonner.

1. a. De **−128 à 127**. Le total reste 256 valeurs :
$127 + 128 + 1 = 256$
2. Les 256 combinaisons sont réparties entre négatifs, zéro et positifs.
3. b. **−128** : en **complément à 2**, le bit de poids fort porte une puissance **négative**.
4. c. L'addition fonctionne **sans cas particulier** : le **même circuit** additionne des nombres positifs et négatifs, sans traiter le signe à part.
5. d. On obtient **−128** : le compteur **repart** à l'autre extrémité de la plage. C'est un **débordement**.
6. e. Parce que le résultat est **parfaitement déterministe** : il découle de la représentation choisie et du nombre de bits disponibles. Le défaut est dans le **programme** qui n'a pas vérifié ses bornes, pas dans la machine.

3 Codage des caractères. Répondre et justifier.

1. a. Sur **7 bits**, soit **128** caractères.
$2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 128$
2. b. Les caractères **ASCII** y conservent leur code, sur **un seul octet**. Un fichier purement ASCII est donc déjà de l'**UTF-8** valide.
3. c. **Non**. Les caractères accentués occupent **deux octets** en **UTF-8**, soit 13 octets au total.
$7 + 3 \times 2 = 13$
4. d. On peut **couper au milieu** d'un caractère codé sur plusieurs octets. La séquence obtenue n'est plus décodable, ce qui produit les caractères illisibles qu'on voit apparaître dans certains logiciels.

4 Les flottants. Analyser.

1. a. Pour la même raison qu'un **tiers** n'en a pas en base 10 : l'écriture d'une fraction n'est finie que si son dénominateur se décompose avec les **facteurs premiers de la base**. Dix vaut 2 × 5, et le facteur 5 ne se retrouve pas en base 2.
2. b. Une **approximation** : un signe, un **exposant** et une **mantisse** sur un nombre **fini** de bits. La valeur stockée est la plus proche représentable, pas la valeur exacte.
3. c. Parce qu'aucun des trois nombres n'est stocké exactement. L'égalité porte sur des **approximations**, et deux chemins de calcul différents n'aboutissent pas nécessairement au même arrondi.
4. d. En comparant l'**écart absolu** à un **seuil** choisi : vérifier que la différence est inférieure à une tolérance, jamais tester l'égalité stricte.
5. e. Partout où les erreurs **s'accumulent** ou où le résultat est comparé à un seuil : calculs financiers répétés, simulations sur de nombreuses itérations, conditions d'arrêt d'un algorithme numérique. Une erreur d'arrondi négligeable une fois ne l'est plus après un million d'opérations.

Corrige-toi toi-même. Compte tes points exercice par exercice : c'est ainsi qu'on voit où l'on perd, et ce n'est presque jamais là où on le croit.

Ce qui est évalué	Points	Ce qui fait perdre le point
Convertir entre bases avec le détail	4 pts	Compter les puissances de gauche à droite
Regrouper par quatre bits pour l'hexadécimal	3 pts	Passer par le décimal inutilement
Expliquer le complément à 2 et sa plage	4 pts	Traiter le bit de signe comme une valeur positive
Distinguer caractères et octets en UTF-8	4 pts	Compter un octet par caractère
Expliquer l'approximation des flottants	5 pts	Attribuer l'écart à un bug de la machine