Fiche d'exercices pour la partie « architecture des ordinateurs » du M1 bioinformatique (avec corrections)

Chapitre 1 – architecture des ordinateurs

Thème : les entiers positifs

Exercice 1. Ecrivez le nombre 312 en binaire.

Correction détaillée :

- Les puissances de 2 sont 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, etc.
- La plus grande puissance de 2 inférieure ou égale à 312 est 256. Donc 312=256+a, et on obtient par calcul a=56.
- La plus grande puissance de 2 inférieure ou égale à 56 est 32. Donc 56=32+b, et on obtient par calcul b=24.
- La plus grande puissance de 2 inférieure ou égale à 24 est 16. Donc 24=16+c, et on obtient par calcul c=8.
- La plus grande puissance de 2 inférieure ou égale à 8 est 8. Donc 8=8+d, et on obtient par calcul d=0.
- Comme on arrive à 0, on s'arrête. On a ainsi obtenu 312=256+32+16+8. En réécrivant cette addition avec les puissances de 2, on obtient $312=2^7+2^5+2^4+2^3$. On peut réinsérer les puissances manquantes en les multipliant par 0, et mettre un facteur 1 devant toutes les puissances que l'on avait trouvées. On obtient alors $312=1*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 0*2^0$. On obtient ainsi $312=1011\ 1000_2$.
- La réponse est $1011\ 1000_2$

Exercice 2. Ecrivez le nombre 39 en binaire.

Résultat attendu : $10\ 0111_2$

Exercice 3. Quelle est la valeur décimale du nombre binaire 1 0011 ?

Correction détaillée :

- Il y a cinq bits dans 1 0011, donc on va commencer avec $2^{5-1}=2^4=16$.
- 1 0011 en base 2 vaut par définition $1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 1*16 + 1*2 + 1*1 = 16+2+1=19$
- La réponse est 19

Exercice 4. Quelle est la valeur décimale du nombre binaire 1101 ?

Résultat attendu : 13

Exercice 5. Quel est le plus grand nombre que l'on peut représenter sur 4 bits ? Sur 8 bits ?

Résultat attendu : 15 pour 4 bits, et 255 pour 8 bits.

Thème : les entiers négatifs

Exercice 6. Ecrivez le nombre +20 en complément logique sur 8 bits.

Correction détaillée :

- Comme il s'agit d'un nombre positif, il s'écrit comme un entier positif (donc le complément logique n'intervient pas).
- 20 = 16+4, donc le nombre s'écrit 1 0100. Sur 8 bits, il faut insérer des 0 à gauche, donc on obtient 0001 0100.
- La réponse est 0001 0100

Exercice 7. Ecrivez le nombre -23 en complément logique sur 8 bits.

Correction détaillée :

- Il s'agit d'un nombre négatif. On commence par écrire +23 sur 8 bits.
- $23=16+4+2+1=1\ 0111_2=0001\ 0111_2$ sur 8 bits
- Comme on veut la valeur en complément logique, il suffit maintenant d'inverser tous les bits. On obtient 1110 1000.
- La réponse est 1110 1000

Exercice 8. Ecrivez le nombre -37 en complément logique sur 8 bits.

Résultat attendu : 1101 1010

Exercice 9. Quelle est la valeur décimale du nombre 0000 0111, représenté en complément logique sur 8 bits ?

Correction détaillée :

- Comme le bit le plus à gauche est 0 (dans la représentation sur 8 bits), il s'agit d'un nombre positif. Donc, le complément logique ne va pas intervenir.
- $0000\ 0111_2=4+2+1=7$
- La réponse est +7

Exercice 10. Quelle est la valeur décimale du nombre 1110 1110, représenté en complément logique sur 8 bits ?

Correction détaillée :

- Comme le bit le plus à gauche est 1 (dans la représentation sur 8 bits), il s'agit d'un nombre négatif.
- On inverse tous les bits : on obtient alors 0001 0001
- On calcule la valeur décimale de ce nombre : on obtient 16+1=17
- La réponse est -17

Exercice 11. Quelle est la valeur décimale du nombre 0100 0000, représenté en complément logique sur 8 bits ?

Résultat attendu : +64

Exercice 12. Quelle est la valeur décimale du nombre 1100 0011, représenté en complément logique sur 8 bits ?

Résultat attendu : -60

Exercice 13. Ecrivez le nombre +13 en complément arithmétique sur 8 bits.

Résultat attendu : 0000 1101

Exercice 14. Ecrivez le nombre -13 en complément arithmétique sur 8 bits.

Correction détaillée :

- -13 est un nombre négatif, donc le complément arithmétique va intervenir.
- On commence par écrire le nombre +13 sur 8 bits. On obtient 0000 1101
- On inverse tous les bits. On obtient 1111 0010
- On ajoute 1. On obtient 1111 0011
- La réponse est 1111 0011

Exercice 15. Ecrivez le nombre -14 en complément arithmétique sur 8 bits.

Résultat attendu : 1111 0010

Exercice 16. Quelle est la valeur décimale du nombre 1100 1100, représenté en complément arithmétique sur 8 bits ?

Correction détaillée :

- Comme le bit le plus à gauche (dans la représentation sur 8 bits) est 1, il s'agit d'un nombre négatif. Le complément arithmétique va intervenir.
- On commence par supprimer 1. On obtient 1100 1011. Remarque : pour s'en assurer, on peut vérifier que abcde011+1 = abcde100, donc abcde100-1=abcde011. Il faut bien faire attention à la retenue, qui fonctionne comme sur les nombres en base 10 (et qui se propage donc à gauche dès que l'on dépasse la base, qui est ici 2).
- On inverse ensuite tous les bits. On obtient 0011 0100
- Le nombre associé est 32+16+4=52
- La réponse est -52

Exercice 17. Quelle est la valeur décimale du nombre 1111 0000, représenté en complément arithmétique sur 8 bits ?

Résultat attendu : -16

Exercice 18. Le nombre 0 peut être considéré à la fois comme un nombre positif (on l'écrira +0) ou comme un nombre négatif (on l'écrira -0). Montrez qu'en complément logique sur 8 bits, +0 et -0 ont deux représentations binaires différentes.

Résultat attendu : +0 s'écrit 00000000 et -0 s'écrit 11111111

Exercice 19. Montrez qu'en complément arithmétique sur 8 bits, +0 et -0 ont la même représentation binaire.

Résultat attendu : +0 s'écrit 00000000 et -0 s'écrit 00000000

Thème : les nombres réels

Exercice 20. Ecrivez le nombre +4 au format IEEE 854 simple précision. On rappelle que dans ce format, il y a un bit de signe, 23 bits de mantisse, 8 bits d'exposant, que la mantisse a un bit caché, et que l'exposant est biaisé de 126.

Correction détaillée :

- Le nombre +3 est positif, donc le bit de signe est 0
- Il faut à présent écrire +3 sous la forme $m*2^e$, avec 0.5<=m<1. On obtient successivement : $3=3*2^0=1.5*2^1=0.75*2^2$. On a ainsi m=0.75 (qui est bien compris entre 0.5 et 1) et e=2.
- On va maintenant coder la mantisse. Les puissances négatives de 2 sont 0.5, 0.25, 0.125, …
- La plus grande puissance de 2 inférieure ou égale à 0.75 est 0.5. On a donc 0.75=0.5+a, et on obtient par calcul a=0.25
- La plus grande puissance de 2 inférieure ou égale à 0.25 est 0.25. On a donc 0.25=0.25+b, et on obtient par calcul b=0.
- On s'arrête car on a obtenu 0. La mantisse obtenue est 0,5+0,25, ce qui s'écrit $0*2^0+1*2^{-1}+1*2^{-2}+0*2^{-3}+0*2^{-4}+0*2^{-5}+…=0,1100\ 0…_2$. On ne va pas représenter le bit avant la virgule. La mantisse a un bit caché, donc on ne va pas représenter non plus le premier bit après la virgule, qui est toujours 1 (car la mantisse est toujours supérieure ou égale à 0.5 par définition). La mantisse est donc (1)100…, c'est-à-dire 100 0000 0000 0000 0000 0000 sur 23 bits.
- L'exposant est e=2. Comme il doit être biaisé de 126, il faut lui ajouter 126. On obtient e+126=128. On va maintenant représenter 128 sur 8 bits, ce qui donne $1000\ 0000_2$.
- Le résultat est donc 0 puis 100 0000 0000 0000 0000 0000 puis 1000 0000.
- La réponse est 0100 0000 0000 0000 0000 0000 0000 0000

Exercice 21. Ecrivez le nombre -0.3125 au format IEEE 854 simple précision.

Correction détaillée :

- -0.3125 est un nombre négatif, donc le bit de signe est 1.
- 0.3125 s'écrit $0.3125*2^0=0.625*2^{-1}$. On a donc m=0.625 (ce qui est bien compris entre 0.5 et 1) et e=-1.
- Pour représenter la mantisse, on détermine que 0.625=0.5+0.125, ce qui s'écrit $0*2^0+1*2^{-1}+0*2^{-2}+1*2^{-3}+0*2^{-4}+0*2^{-5}+0*2^{-6}+…=0,1010\ 00…_2$
- Comme la mantisse est à bit caché, ce 0,1010 00… va se représenter (1)0100 0…, c'est-à-dire 0100 0… Sur 23 bits, cela donne 010 0000 0000 0000 0000 0000
- L'exposant est e=-1. Comme il doit être biaisé de 126, il faut lui ajouter 126. On obtient 125. La représentation de 125 sur 8 bits est 0111 1100.
- Le résultat est donc 1 puis 010 0000 0000 0000 0000 0000 puis 0111 1100
- La réponse est 1010 0000 0000 0000 0000 0000 0111 1100

Exercice 22. Quelle est la valeur décimale du nombre 1110 0000 0000 0000 0000 0000 0101 1100, représenté au format IEEE 854 simple précision ?

Correction détaillée :

- Le premier bit de ce nombre est le bit de signe. Comme il s'agit d'un 1, le nombre recherché est négatif.
- Les 23 bits suivants sont 110 0000 0000 0000 0000 0000. En ajoutant les bits cachés et la virgule, on obtient 0,1 110 00000… Cela s'écrit 0,111 en enlevant les 0 inutiles à droite. $0,111=0*2^0+1*2^{-1}+1*2^{-2}+1*2^{-3}=0.5+0.25+0.125=0.875$. On a donc m=0.875 (ce qui est bien compris entre 0.5 et 1)
- Les 8 bits suivants sont l'exposant biaisé. La valeur est 0101 1100. Cela correspond à l'entier biaisé 64+16+8+4=92. Pour obtenir l'exposant non biaisé, il faut retirer 126. On a donc e=92-126=-34.
- Le nombre est donc $-0.875*2^{-34}$. Vu l'exposant (qui est un « grand » nombre négatif), il s'agit d'un nombre extrêmement petit. Il est approximativement égal à $5*10^{-11}$.

Exercice 23. Certains nombres n'ont pas une représentation finie en base 10. C'est par exemple le cas de 1/3, qui s'écrit 0,33333… Toutefois, il se pourrait que 1/3 s'écrive de manière finie dans d'autres bases que la base 10. C'est le cas en base 3, avec 1/3 qui s'écrit $0,1_3$. Dans cet exercice, nous allons considérer le cas de 1/10, qui s'écrit de manière finie en base 10 (puisqu'il s'écrit évidemment 0,1 en base 10). Malheureusement, ce nombre 1/10 n'a pas de représentation finie en base 2. Pour vous en donner une intuition, prenez une calculatrice, et essayez d'écrire 1/10 comme la somme de puissances négatives de 2, jusqu'à ce que vous vous lassiez (vous n'obtiendrez jamais 0).

Résultat attendu : 1/10=0.0625+0.03125+0.00390625+…=0,0001 1001 …

Exercice 24. Considérons un programme informatique avec un compteur qui part de 0. Si vous ajoutez la valeur 0,1 dix fois de suite, on devrait obtenir le résultat 1. Or, le programme informatique n'obtient pas 1 mais une valeur légèrement inférieure à 1. Expliquez pourquoi. Vous pouvez vous inspirer du fait que si on ajoute 0,33333 trois fois de suite en base 10, on n'obtient pas le résultat 1, mais 0.99999. Ce comportement peut être la source de bugs dans les programmes.

Correction détaillée : comme il n'est pas possible de représenter 1/10 en base 2, la représentation approchée de 1/10 est légèrement inférieure à la vraie valeur. Ainsi, en l'ajoutant un certain nombre de fois (et ici, dix fois), le résultat approché obtenu devient de plus en plus différent du résultat théorique qui est 1. Le programme va donc effectuer la boucle onze fois au lieu de dix.

Thème : les circuits logiques

Exercice 25. Dessinez la table de vérité des fonctions logiques OR, AND, XOR, NOR, NAND.

Résultat attendu :

OR	A=0	A=1	AND	A=0	A=1	XOR	A=0	A=1	NOR	A=0	A=1	NAND	A=0	A=1
B=0	0	1	B=0	0	0	B=0	0	1	B=0	1	0	B=0	1	0
B=1	1	1	B=1	0	1	B=1	1	0	B=1	0	0	B=1	0	0

Exercice 26. On souhaite réaliser une alarme qui sonne lorsque l'on appuie sur certains boutons. Si le bouton rouge est appuyé, l'alarme sonne. Si les boutons vert et bleu sont tous les deux appuyés en même temps, alors l'alarme sonne aussi. Dessinez la table de vérité de l'alarme.

Correction détaillée :

- On commence par identifier les entrées et les sorties du système. La sortie est l'alarme : 0 = elle ne sonne pas, et 1 = elle sonne. Les entrées sont les trois boutons. Comme il y a 3 entrées, on aura 2^3=8 lignes dans la table de vérité (sans compter l'entête). On inscrit toutes les combinaisons possibles des entrées dans le tableau, ce qui donne le tableau ci-dessous (encore incomplet).

Bouton rouge	Bouton vert	Bouton bleu	Alarme
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

- La phrase sur le bouton rouge dit que lorsque le bouton rouge vaut 1, l'alarme vaut 1, quelque soit les autres valeurs. On peut donc remplir le tableau comme représenté ci-dessous.

Bouton rouge	Bouton vert	Bouton bleu	Alarme
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

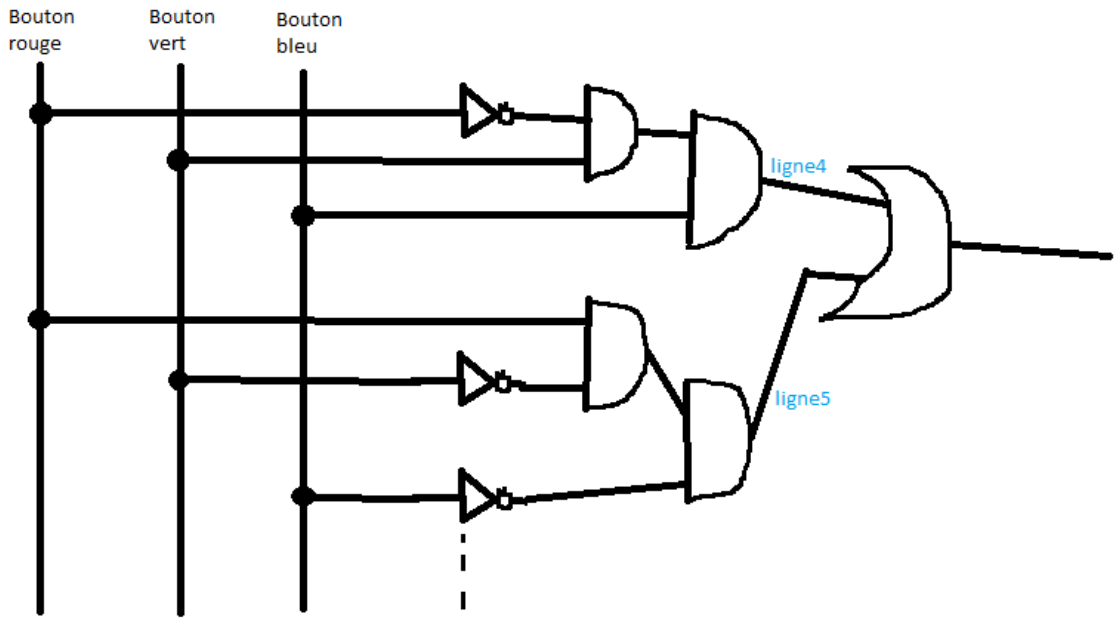
- La phrase sur les boutons vert et bleu indique que l'alarme sonne si le bouton vert vaut 1 et le bouton bleu vaut 1, mais pas dans les autres cas. On peut donc remplir le tableau comme représenté ci-dessous.

Bouton rouge	Bouton vert	Bouton bleu	Alarme
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Exercice 27. Dessinez le circuit logique correspondant à cette alarme.

Correction détaillée (partielle) :

- Tout d'abord, il faut exprimer la table de vérité comme un fonction logique. Pour cela, le moyen le plus simple (mais pas toujours le plus rapide) est de ne regarder que les lignes où l'alarme vaut 1. On a cinq lignes de ce type. La fonction logique va donc être : alarme = ligne 4 ou bien ligne 5 ou bien ligne 6 ou bien ligne 7 ou bien ligne 8. Cela s'écrit avec un OR de la manière suivante : alarme = ligne4 OR ligne5 OR ligne6 OR ligne7 OR ligne8.
- Pour écrire une ligne comme une fonction logique, il suffit de remarquer que la ligne correspond à plusieurs propriétés qui doivent être vraies en même temps. Par exemple, ligne4 = bouton rouge vaut 0 et bouton vert vaut 1 et bouton bleu vaut 1. Cela s'écrit avec un AND de la manière suivante : ligne4=NOT(rouge) AND vert AND bleu, ligne5=rouge AND NOT(vert) AND NOT(bleu)
- On obtient donc au final : alarme = (NOT(rouge) AND vert AND bleu) OR (rouge AND NOT(vert) AND NOT(bleu)) OR …
- Une fois qu'on a la fonction logique, il suffit de la dessiner, comme indiqué ci-dessous.



Thème : les circuits séquentiels

Exercice 28. Un circuit séquentiel est défini comme un circuit logique avec une notion de temps et/ou une notion de stockage. Mais, on aurait pu définir un circuit séquentiel comme un circuit logique avec une notion de temps. Expliquez pourquoi.

Correction détaillée : Comme il est possible d'implémenter du stockage au moyen d'une bascule RS, et que cette bascule n'utilise qu'une notion de temps, alors tout circuit séquentiel peut être défini comme un circuit logique avec uniquement une notion de temps (le stockage pouvant s'implémenter grâce à cela).

Exercice 29. Supposons une bascule RS dans lequel l'état de la mémoire est Q=0, et dans lequel on n'utilise ni la commande Reset, ni la commande Set. Montrez que le nouvel état de la mémoire reste Q=0.

Correction détaillée :

- On a donc Q=0, not(Q)=1, S=0, R=0.
- En entrée du premier NOR, on a S=0 et not(Q)=1. Donc la sortie de ce NOR est 0, ce qui correspond bien à Q=0 (qui n'a pas changé).
- En entrée du deuxième NOR, on a Q=0 et R=0. Donc la sortie de ce NOR est 1, ce qui correspond bien à Q=1 (qui n'a pas changé).

Exercice 30. Supposons une bascule RS dans lequel l'état de la mémoire est Q=1, et dans lequel on n'utilise ni la commande Reset, ni la commande Set. Montrez que le nouvel état de la mémoire reste Q=1.

Résultat attendu : le nouvel état de Q doit rapidement converger vers 1.

Exercice 31. Supposons une bascule RS dans lequel l'état de la mémoire est Q=0, et dans lequel on utilise la commande Set. Montrez que le nouvel état de la mémoire est Q=1.

Correction détaillée :

- Initialement, on a Q=0, not(Q)=1, R=0, S=1.
- En entrée du premier NOR, on a R=0 et not(Q)=1. La sortie est donc Q=0, qui n'a pas changé.
- En entrée du deuxième NOR, on a Q=0 et S=1. La sortie est donc not(Q)=0, qui a changé.
- Comme not(Q) a changé, il faut recalculer la sortie du premier NOR. L'entrée du premier NOR est à présent R=0 et not(Q)=0. La sortie est donc Q=1, qui a changé.
- Comme Q a changé, il faut recalculer la sortie du deuxième NOR. L'entrée du deuxième NOR est à présent S=1 et Q=1. La sortie est donc not(Q)=0, qui n'a pas changé.
- On obtient bien Q=1.

Exercice 32. Supposons une bascule RS dans lequel l'état de la mémoire est Q=1, et dans lequel on utilise la commande Reset. Montrez que le nouvel état de la mémoire est Q=0.

Résultat attendu : la variable Q doit rapidement converger vers 0.

Thème : la carte mère

Exercice 33. Dans la photographie de carte mère 1/5 (dans les slides du cours), repérez où sont les contrôleur de bus nord et de bus sud.

Correction détaillée : le contrôleur de bus nord se trouve aux intersections des droites ((2),(8)) et ((4),(1)). Il est situé sous un dispositif permettant de le refroidir. Le contrôleur de bus sud se trouve au milieu du segment ((3),(5)), à droite de (6).

Exercice 34. Dans la photographie de carte mère 2/5 (dans les slides du cours), que représente la légende « chipset » ?

Résultat attendu : il s'agit du contrôleur de bus nord.

Exercice 35. Dans la photographie de carte mère 3/5 (dans les slides du cours), où est le processeur ?

Résultat attendu : il est sous le ventilateur.

Exercice 36. Dans la photographie de carte mère 3/5 (dans les slides du cours), expliquez le fait que les nappes IDE (qui sont donc des bus) ne sont pas des câbles cylindriques mais plats.

Correction détaillée : Les bus sont des ensembles de fils pouvant transmettre des données. Par exemple, pour transmettre 16 bits de données en parallèle, il faut 16 fils. Les nappes sont ainsi constituées de plusieurs fils les uns à côté des autres, ce qui leur donne cet aspect plat.

Exercice 37. Dans la photographie de carte mère 4/5 (dans les slides du cours), indiquez où se trouvent : le processeur, la mémoire et le bus AGP.

Résultat attendu : le processeur se trouve à gauche, la mémoire au-dessus, et le bus AGP vers le centre (à droite du processeur).

Exercice 38. Dans les photographies de toutes les cartes mères (dans les slides du cours), pourquoi les ports des périphériques se trouvent souvent sur le bord de la carte mère ?

Correction détaillée : Cela permet de faciliter l'interconnexion entre ces ports et le boîtier, car les périphériques correspondants seront connectés ensuite au boîtier. Les périphériques qui sont branchés directement sur la carte mère, et qui ne sont pas supposés être ajoutés/enlevés régulièrement, sont branchés sur les ports au milieu de la carte mère (comme par exemple : la carte son, la carte réseau, la carte graphique, la mémoire, etc.).

Exercice 39. Lorsque l'on souhaite se souvenir d'informations utiles d'un document, on fait souvent une fiche de synthèse, qui contient des éléments importants du document. En utilisant cette analogie, détaillez les avantages et inconvénients de la mémoire cache par rapport à la mémoire vive.

Correction détaillée :

- Dans cette analogie, on peut considérer que le document (complet) est dans la mémoire vive, et que la fiche de synthèse est la mémoire cache.
- Avantages de la mémoire cache (= avantages de la fiche de synthèse) : Lorsque l'on cherche une information, il est beaucoup plus rapide de la trouver sur la fiche de synthèse que dans le livre complet. L'accès aux données utiles est donc très rapide.
- Inconvénients de la mémoire cache : la mémoire cache a une taille limitée (par exemple, le recto d'une feuille A4 pour une fiche de synthèse). On ne peut donc pas stocker beaucoup d'information, et il est nécessaire de faire du tri dans les informations. S'il est difficile de savoir ce qui est utile dans un document, on risque aussi de remplir la fiche de synthèse avec des données peu utiles, et on ne pourra pas gagner de temps en essayant d'utiliser cette fiche de synthèse lorsque l'on cherche des données utiles (car ces données ne seront pas dans la fiche de synthèse).

Exercice 40. Reprenons la même analogie, mais imaginons à présent qu'en plus de la fiche de synthèse, on s'autorise à surligner certains passages du document. Qu'est-ce qui représente la mémoire cache de niveau 1 ? Qu'est-ce qui représente la mémoire cache de niveau 2 ?

Correction détaillée :

- La mémoire cache de niveau 1 est rapide mais petite. La mémoire cache de niveau 2 est un peu plus lente, et un peu plus grosse. La mémoire vive est très lente, mais très grosse.
- La mémoire cache de niveau 1 reste la fiche de synthèse : on ne peut stocker qu'un recto de page A4, et on y accède très vite.
- La mémoire cache de niveau 2 est l'ensemble des informations surlignées. On peut en surligner beaucoup, mais il faut feuilleter le document pour savoir ce qui a été surligné, ce qui prend du temps (mais moins que de relire tout le document).

Thème : l'optimisation du code

Exercice 41. Lorsque l'on multiplie un nombre entier n par 101 en base 10, plutôt que de poser la multiplication, il est beaucoup plus rapide de calculer 100*n+n, car le 100*n se calcule rapidement en ajoutant deux 0 à droite de la représentation en base 10 de n. Montrez que lorsque l'on multiplie un nombre entier n par 33 en base 2, plutôt que de poser la multiplication, il est beaucoup plus rapide de calculer 32*n+n. Vous expliquerez comment calculer rapidement 32*n.

Résultat attendu : 33*n = 2*2*2*2*2*n+n, ce qui nécessite donc cinq multiplications par 2 (très rapides en base 2) et une addition.

Exercice 42. Supposons que l'on cherche à savoir si la somme de toutes les cases paires d'un tableau est inférieure, supérieure ou égale à la somme de toutes les cases impaires d'un tableau. Par exemple, pour le tableau (3,8,2,9,10,3,7), on cherche à comparer 3+2+10+7=22 à 8+9+3=20. Considérons l'algorithme suivant. Montrez sur un exemple que cet algorithme fonctionne.

```
sommeIndicesPairs := 0
sommeIndicesImpairs := 0
Pour i de 1 à n faire
```

```
  Si i est pair alors
    ajouter t[i] à sommeIndicesPairs
  Sinon
    ajouter t[i] à sommeIndicesImpairs
  Finsi
Finpour
```

Correction détaillée :

- On va faire la trace de l'algorithme, c'est-à-dire indiquer la valeur des variables à chaque instruction. On va partir de l'exemple de l'énoncé.
- Initialement, sommeIndicesPairs=0 et sommeIndicesImpairs=0.
- Quand i=1, comme 1 est impair, on va ajouter 3 à sommeIndicesImpairs ; sommeIndicesPairs=0 et sommeIndicesImpairs=3
- Quand i=2, comme 2 est pair, on va ajouter 8 à sommeIndicesPairs ; sommeIndicesPairs=8 et sommeIndicesImpairs=3
- Quand i=3, on aura sommeIndicesPairs=8 et sommeIndicesImpairs=5
- Quand i=4, on aura sommeIndicesPairs=17 et sommeIndicesImpairs=5
- Quand i=5, on aura sommeIndicesPairs=17 et sommeIndicesImpairs=15
- Quand i=6, on aura sommeIndicesPairs=20 et sommeIndicesImpairs=15
- Quand i=7, on aura sommeIndicesPairs=20 et sommeIndicesImpairs=22
- L'algorithme termine avec les bonnes valeurs de sommeIndicesPairs et sommeIndicesImpairs

Exercice 43. Reprenons le même algorithme. Supposons que l'on peut charger dans une mémoire rapide (soit dans des registres du processeur, soit dans la mémoire cache) les variables sommeIndicesPairs et sommeIndicesImpairs. Montrez comment fonctionne cet algorithme sur un exemple, en indiquant les accès « hits » et les accès « miss » dans la mémoire rapide. Notez que chaque accès « hit » permet de gagner du temps, et que chaque accès « miss » en fait perdre.

Résultat attendu : Il n'y a aucun changement sur le résultat du programme. Tous les accès à sommeIndicesPairs et sommeIndicesImpairs sont des « hits » (sauf l'initialisation de ces deux variables qui sont des « miss »)

Exercice 44. Reprenons le même algorithme. Supposons à présent que l'on ne peut charger dans une mémoire rapide (qu'il s'agisse d'un registre du processeur ou de la mémoire cache) qu'une seule des deux variables. Montrez comment fonctionne cet algorithme sur l'exemple précédent, en indiquant les accès « hits » et les accès « miss » dans la mémoire rapide.

Résultat attendu :

- Si vous avez considéré que l'une des deux variables était en cache, et que cette variable ne changeait pas, une itération sur deux causera un « hit » et une itération sur deux causera un « miss ».
- Si vous avez considéré que c'était toujours la dernière variable utilisée qui était en cache (ce qui correspond à ce que les processeurs réels font), alors à chaque itération, vous aurez un « miss » (car c'est toujours la mauvaise variable qui est chargée, étant donné que les itérations alternent l'utilisation de la variable).

Exercice 45. Reprenons le même algorithme, et gardons l'hypothèse que la mémoire rapide ne peut stocker qu'une seule variable. Réécrivez l'algorithme pour profiter au mieux de la mémoire cache.

Résultat attendu : il faut réécrire la boucle de 1 à n pour qu'elle traite d'abord tous les indices pairs, puis tous les indices impairs. Ainsi, vous aurez un « hit » à chaque itération, sauf à la première itération de chaque boucle (ou de la deuxième boucle seulement, selon vos hypothèses). Pour séparer les indices, vous pouvez soit faire deux boucles (avec un pas de 2), soit faire une boucle qui utilise deux cases à chaque itération.

Thème : les limites d'un ordinateur

Exercice 46. Pourquoi la mémoire vive d'un ordinateur est-elle limitée ?

Correction détaillée : La mémoire vive est construite à l'aide de bascules RS. Chaque bascule nécessite un circuit logique, qui prend une (petite) place. Chaque barète de mémoire vive comprend des milliards de ces bascules (une barète d'1 Go de RAM contient 8*2^30 bascules RS, soit un peu plus de 8 milliards). Mais la carte mère ne peut contenir qu'un nombre limité de barètes (généralement 2 ou 4), donc un nombre limité (bien que grand) de bascules. Une autre contrainte sera l'adressage : chaque bascule RS doit pouvoir être adressée, et doit donc avoir un numéro. Le nombre de bits utilisé pour représenter ce numéro est une limite matérielle à la mémoire vive disponible dans un ordinateur.

Exercice 47. Indiquez les circuits (principaux) qui sont franchis lorsque vous imprimez un fichier depuis votre ordinateur.

Correction détaillée :

- Disque dur => contrôleur du bus sud => mémoire => processeur => contrôleur du bus sud => imprimante

Exercice 48. Que se passe-t'il si deux périphériques qui sont connectés sur le même bus, souhaitent transmettre en même temps des données au processeur ? Quel est l'impact sur la vitesse d'exécution ?

Correction détaillée : le bus ne permet pas un accès simultané du bus à plusieurs périphériques, donc le contrôleur du bus arbitrera les communications. L'un des périphériques enverra ses données en premier pendant que l'autre attend, puis le deuxième enverra ses données. Cette attente cause du délai.

Exercice 49. Quels sont les avantages et inconvénients d'un processeur générique par rapport à un processeur graphique ?

Correction détaillée :

- Un processeur générique est capable d'exécuter des instructions très variées. Toutefois, cette capacité ralentit sa vitesse de traitement (d'une part il y a une étape de « décodage » de l'instruction qui est d'autant plus lente que le processeur comprend un grand nombre d'instructions, et d'autre part, la vitesse du processeur dépend de la vitesse de traitement de la plus complexe des instructions, afin de garder une synchronisation dans l'exécution des instructions).
- Un processeur graphique ne peut exécuter que certaines instructions (graphiques) très spécifiques (comme par exemple calculer un angle ou une distance). Par contre, il peut les exécuter extrêmement rapidement, car il est optimisé pour ce traitement. L'optimisation peut se faire avec un traitement en parallèle des instructions sur plusieurs « sous-processeurs graphiques ».

Chapitre 2 – programmation parallèle

Thème : la loi d'Amdhal

Exercice 50. Considérons un traitement qui peut être parallélisé à 20%. Si l'exécution de ce traitement dure une heure trente sur un unique processeur, quelle est la vitesse que l'on peut attendre sur deux processeurs ?

Correction détaillée : La loi d'Amdhal nous indique que l'accélération en parallélisant est 1/(1-p+p/s). Comme nous avons p=0.2 et s=2, on a p/s=0.1, 1-p+p/s=1-0.2+0.1=0.9, soit une accélération de 1/0.9=1.11 (environ). La durée sur un processeur était d'une heure trente, soit 90 minutes. La durée sur deux processeurs sera donc 90/1.11=81.08 minutes (environ). On aura donc gagné environ 9 minutes.

Exercice 51. Considérons un traitement parallélisable à 50% qui prend trois heures sur deux processeurs. Quelle est la vitesse que l'on peut attendre sur quatre processeurs ?

Correction détaillée : Nous avons p=0.5 et s=2. Le gain entre le temps pour un processeur et pour deux processeurs est 1/(1-p+p/s)=1/(1-0.5+0.25)=1/0.75=1.333 (environ). Le gain entre le temps pour un processeur et le temps pour quatre processeurs est 1/(1-p+p/s)=1/(1-0.5+0.125)=1/0.625=1.6. Donc le gain entre deux processeurs et quatre processeurs est 1.6/1.333=1.20 (environ), soit 20% de gain de temps. Au lieu de trois heures (donc 180 minutes), le temps va être réduit de 20%, ce qui nous donne : 180/1.20=150=deux heures trente.

Thème : les processus et les accès concurrents

Exercice 52. Considérons deux processus : l'un qui écrit la chaîne de caractères « 11111111\n » à l'écran, et l'autre qui écrit la chaîne de caractères « 22222222\n » à l'écran. Présentez des scénarios d'exécution qui produisent :

- « 11111111\n » puis « 22222222\n »
- « 111122222222\n » puis « 1111\n »
- « 1122211111122222\n » puis « \n »

Correction détaillée :

- La première séquence correspond au processus 1 qui s'exécute en premier et complètement (résultat : « 11111111\n »), puis le processus 2 qui s'exécute ensuite et complètement (résultat « 22222222\n »).
- La deuxième séquence correspond au processus 1 qui commence à s'exécuter (résultat « 1111 » puis le processus 2 qui s'exécute ensuite et complètement (résultat « 22222222\n »), puis le processus 1 qui reprend la main et termine son exécution (résultat « 1111\n »).
- La troisième séquence correspond au processus 1 qui commence à s'exécuter (résultat « 11 »), puis le processus 2 qui commence à s'exécuter (résultat « 222 »), puis le processus 1 qui reprend l'exécution (résultat « 111111 »), puis le processus 2 qui continue et termine (résultat « 22222\n »), puis le processus 1 qui termine (résultat « \n »). Remarque : il est aussi possible que le premier « \n » soit écrit par le processus 1 et le deuxième « \n » soit écrit par le processus 2.

Exercice 53. Dans le dîner des philosophes, expliquez ce qu'est une situation d'interblocage (aussi appelé « *deadlock* » en anglais).

Correction détaillée : si chaque processus prend la fourchette gauche, alors les processus seront tous en attente de la fourchette droite qui ne se libèrera jamais, et donc ils seront tous bloqués.

Exercice 54. Dans le dîner des philosophes, décrivez une situation conduisant à la famine de l'un (uniquement) des processus.

Correction détaillée : Considérons quatre processus A, B et C. Considérons que B attende que ses deux fourchettes soient libres pour pouvoir manger. Ainsi, B devra attendre que ni A ni C ne mange pour pouvoir manger (on peut d'ailleurs remarquer que A et C ne peuvent pas manger en même temps). Mais si A et C alternent le repas, alors B n'aura jamais la capacité de manger, et sera donc dans une situation de famine. Remarque : l'énoncé visait à interdire l'argumentaire de l'interblocage, en limitant la famine à un processus. En effet, s'il y a interblocage, tous les processus seront en famine.

Exercice 55. Dans le dîner des philosophes, montrez que la solution suivante n'est pas correcte :

```
Tantque vrai faire
   Attendre fourchette gauche
   Prendre fourchette gauche
   Attendre fourchette droite
   Prendre fourchette droite
   Manger
Fintantque
```

Correction détaillée : Il s'agit d'une implémentation très naïve du dîner des philosophes, avec plusieurs erreurs. En effet :

- Cette solution peut conduire à un interblocage si tous les processus sont bloqués juste après « prendre fourchette gauche ».
- Les accès aux variables partagées (les fourchettes) n'étant pas protégés, il est possible que deux processus prennent la même fourchette. C'est par exemple le cas avec le séquencement suivant : le processus A s'exécute, trouve la fourchette gauche libre, la prend, puis regarde la fourchette droite et la trouve libre. Mais, il n'a pas le temps de la prendre car il est interrompu. Le processus B s'exécute et trouve la fourchette gauche libre. Il la prend, puis il est interrompu. Puis, le processus A continue là où il était interrompu, et prend la fourchette droite. Or, la fourchette droite de A est la fourchette gauche de B. La même fourchette a été prise deux fois, ce qui est interdit.

Exercice 56. Dans le dîner des philosophes, commentez la solution suivante :

```
Tantque vrai faire
   Si processus=1 alors
      Semaphore-wait(fourchette gauche)
      Prendre fourchette gauche
      Semaphore-wait(fourchette droite)
      Prendre fourchette droite
   Sinon
      Semaphore-wait(fourchette droite)
      Prendre fourchette droite
      Semaphore-wait(fourchette gauche)
      Prendre fourchette gauche
   Finsi
   Manger
   Semaphore-signal(fourchette gauche)
   Semaphore-signal(fourchette droite)
Fintantque
```

Correction détaillée :

- Concernant l'accès aux variables partagées, qui sont les fourchettes, tout semble correct : les opérations « prendre » et « manger » sont bien protégées par des sémaphores, et les sémaphores sont bien libérées.
- On peut se demander s'il peut y avoir une solution d'interblocage, dans laquelle tous les processus sont en attente des autres. Grâce au cas particulier du processus 1, qui prend sa fourchette droite puis gauche alors que les autres prennent la fourchette gauche puis droite, il n'est plus possible que tous les processus soient bloqués. La situation la plus critique serait celle où le processus 1 mange (par exemple), et tous les autres processus sont bloqués en attendant leur fourchette gauche. Mais, la situation se résolverait dès que le processus 1 aurait fini de manger, car le processus *n* pourrait alors prendre sa fourchette gauche puis manger, puis libérer la fourchette gauche du processus *n*-1, etc. La solution proposée ne cause donc pas d'interblocage, mais ne semble pas idéale non plus, dans le sens où elle peut générer un grand ralentissement du système (cas critique qui vient d'être décrit).

Exercice 57. Considérons un jeu d'échec dans lequel un joueur humain joue contre un ordinateur. Après chaque coup de l'humain, le programme crée un fichier appelé « coup.dat » et y enregistre le coup. Un processus dédié à l'intelligence artificielle attend l'existence d'un fichier « coup.dat », ouvre le fichier « coup.dat », récupère le coup de l'humain, ferme le fichier, l'efface, calcule le (meilleur) coup à jouer, recrée le fichier « coup.dat », y enregistre le coup et ferme le fichier. Le programme principal attend l'existence du fichier, ouvre le fichier, récupère le coup joué par l'intelligence artificielle, efface le fichier, et affiche le coup à l'écran. Montrez que cette implémentation n'est pas correcte.

Correction détaillée : Cette implémentation a plusieurs problèmes :

- La communication par fichiers entre processus n'est pas idéale dans cet exemple, car un utilisateur mal intentionné pourrait faire croire à l'ordinateur que l'IA a joué son coup en inscrivant un coup factice dans le fichier « coup.dat ».
- En créant/détruisant des fichiers « coup.dat » rapidement, un utilisateur mal intentionné pourrait probablement causer un comportement inattendu du programme principal (comme attendre le résultat de l'intelligence artificielle alors qu'elle a déjà joué son coup).
- Le problème principal est lié à une mauvaise gestion de la variable partagée, qui est ici le fichier. En effet, le processus qui écrit dans le fichier peut être interrompu entre la création du fichier et l'écriture de la position, ce qui fait que le processus lecteur verrait un fichier qui ne contient aucune information. Il faut absolument que « création du fichier » et « écriture dans le fichier » soient dans une section critique, ainsi que « ouverture du fichier » et « lecture dans le fichier », afin que ces opérations ne soient pas interrompues.

Exercice 58. Reprenons l'exemple précédent. Est-ce que mettre un délai d'une seconde entre l'ouverture du fichier et la lecture fonctionnerait ?

Correction détaillée : Le problème de fichier vide serait moins fréquent, car pendant la seconde d'attente entre l'ouverture et la lecture, il est probable que le processus écrivain ait le temps d'écrire. Mais, cette solution ne fonctionne que si le processus sur lequel l'écriture se fait a pu être ordonnancé pendant cette seconde, et la solution continuerait de dysfonctionner si l'ordinateur est trop chargé. Ce n'est donc pas une bonne solution.

Exercice 59. Reprenons l'exemple précédent. Considérons maintenant que chaque processus crée un fichier « coup.tmp » pour y enregistrer le coup, puis renomme le fichier en « coup.dat ». Cette solution fonctionne à présent (même si elle n'est pas idéale). Pourquoi ?

Correction détaillée : Pour que la solution fonctionne, il faut que l'appel à la fonction de renommage d'un fichier soit atomique. C'est le cas sous Linux (comme toutes les opérations « uniques » d'accès à un fichier). L'exemple précédent ne fonctionnait pas car il y avait deux opérations : une opération de création d'un fichier vide, et une opération d'écriture pour les coordonnées du coup.