

M1.L1 : Série d'exercices sur la représentation de l'information [Solutions]

1 Nombre de bits

1. Les mois d'une année = 12 donc requiert au minimum 16 combinaisons, c'est-à-dire 4 bits.
2. Les jours d'un mois = max 31 donc requiert au minimum 32 combinaisons, c'est-à-dire 5 bits.
3. L'ensemble des symboles utilisés pour les nombres romain jusqu' mille =I,V,X,L,C,D,M = 7 donc requiert au minimum 8 combinaisons, c'est-à-dire 3 bits.
4. Les d'étudiants l'EPFL (supposons 11'000). Il faut utiliser quelques puissances de 2 bien connues. Par exemple 2^{10} vaut 1024, donc 16×2^{10} va suffire, c'est-à-dire $2^{(4+10)} = 2^{14}$. C'est à dire 14 bits.
5. Chaque habitant de la planète : environ 7'653'000'000 le 28.09.2018. Il faut utiliser quelques puissances de 2 bien connues. Par exemple 2^{30} vaut un peu plus que 10^9 , donc 8×2^{30} va suffire, c'est-à-dire $2^{(3+30)} = 2^{33}$. C'est-à-dire 33 bits.

Méthode : Comme on utilise des éléments binaires pour représenter les combinaisons distinctes, le nombre de combinaisons pour n bits est 2^n .

Pour cet exercice on fait l'inverse, on cherche le nombre de bits n pour pouvoir représenter au moins K combinaisons distinctes. Si K est une puissance entière de 2, on obtient directement cette puissance entière n en calculant le log de base 2 de K : $\log_2(K) = \log_2(2^n) = n \log_2(2) = n$.

Si par contre K n'est pas une puissance entière de 2, on peut malgré tout exprimer K comme une puissance m de 2, avec m nombre réel compris entre deux entiers consécutifs. La valeur de m est obtenue comme précédemment en prenant le log de base 2 de K. Le nombre de bits recherché n est l'entier immédiatement supérieur à m.

2 Représentation des entiers naturels et décimaux positifs

1. Les bits à droite de la virgule porte des puissances négative de 2 en commençant par -1, etc...
 - 101 donne $2^2 + 1 = 5$,
 - 10.1 donne $2 + 2^{-1} = 2.5$,
 - 1.01 donne $1 + 2^{-2} = 1.25$,
 - 0.101 donne $2^{-1} + 2^{-3} = 0.5 + 0.125 = 0.625$.
2. Remarquez que pour passer d'un nombre au suivant on l'a divisé par la base (i.e. par 2^1), ce qui revient à une multiplication par 2^{-1} . Cela correspond un décalage d'un cran vers la gauche de la virgule. On aurait pu aussi bien diviser la valeur décimale déjà obtenue par deux à chaque fois plutôt que de recalculer à partir de la somme des puissances de 2.
3. on observe que 16 vaut 2^4 . Multiplier par 16 revient à décaler la virgule de 4 crans vers la droite. Le résultat est l'entier 1101 qui est plus facile à évaluer et vaut treize.

3 Domaine couvert des entiers positifs et négatifs, dépassement de capacité et overflow

1. Conversions : motifs binaires sur 8 bits

(a) 00000110 est positif, de valeur $4 + 2 = 6$.

11111001 est négatif car son bit de poids fort est 1. Pour connaître sa valeur absolue, on calcule son opposé en prenant son complément 2, c'est-à-dire le complément 1 auquel on ajoute 1.

11111001

00000110 Cp à 1

00000001 +1

00000111 ce qui vaut 7, donc le motif initial représente -7.

10000110 est négatif car son bit de poids fort est 1. Pour connaître sa valeur absolue, on calcule son opposé en prenant son complément 2, c'est-à-dire le complément 1 auquel on ajoute 1.

10000110

01111001 Cp à 1

00000001 +1

01111010 ce qui vaut $2 + 8 + 16 + 32 + 64 = 122$, donc le motif initial représente -122.

(b) $0 = 00000000$.

-12 calculer d'abord 12 en binaire puis prendre son complément 2

00001100 12

11110011 Cp à 1

00000001 +1

11110100 Cp à 2 = représentation de -12.

-1 calculer d'abord 1 en binaire puis prendre son complément 2

00000001

11111110 Cp à 1

00000001 +1

11111111 Cp à 2 = représentation de -1.

$127_{10} = 01111111$ = maximum des nombres positifs pour 8 bits.

$-128_{10} = 10000000$ = minimum des nombres négatifs pour 8 bits. Ce nombre n'a pas de représentation de son opposé (+128) ou plutôt elle est égale à lui-même.

(c) Conclusion : le domaine n'est pas symétrique. Il faut surveiller le cas singulier où on demande l'opposé du min négatif, car cela donnerait un résultat incorrect.

2. (a) La valeur 64 est représentable avec ce motif binaire : 01000000.

01000000	64
01000000	+64
10000000	donne -128.

(b) Il s'agit d'une violation du domaine couvert (overflow), l'addition de 2 nombres positifs donne un nombre négatif, ce qui est incorrect.

(c) Le dépassement de capacité apparaît chaque fois qu'on ajoute un nombre positif et un nombre négatif et que le résultat est positif.

(d) Une retenue est perdue au-delà du bit de signe. Le résultat est cependant correct.

4 Bases 8 (octal) et 16 (hexadécimal)

1. 110001011

2. 1100101011111110

3. 271_8 ; 212_8 ; $B9_{16}$; $8A_{16}$

4. La méthode rapide consiste à passer d'abord en binaire puis à réorganiser les groupes de bits selon la base de destination. Chaque est ensuite traduit par son symbole.

5 Représentation binaire des entiers et multiplication égyptienne

Voici nos deux colonnes :

1	53
2	106
4	212
8	424
16	848
32	1696

La décomposition de 37 en une somme de puissances de deux est : $37 = 1 \cdot 32 + 1 \cdot 4 + 1 \cdot 1$

Il reste 3 termes à additionner pour obtenir un résultat de : 1961

6 Utilisation d'une représentation à virgule fixe pour représenter la mesure d'un thermomètre numérique familial

1. avec un octet, l'intervalle des entiers naturel couvre l'intervalle [0, 255]. L'erreur absolue est de 1 unité, c'est-à-dire la distance séparant deux valeurs successives.

2. virgule fixe sans décalage : on commence à zéro. Il faut prévoir suffisamment de bits pour que la partie entière couvre l'intervalle [35, 43], c'est-à-dire au moins 6 bits pour un intervalle [0, 63]. Il en reste deux pour la partie fractionnaire. L'erreur absolue maximum est donnée par le poids faible de la partie fractionnaire, c'est-à-dire $2^{-2} = 0,25$ degré.

3. virgule fixe avec décalage : on commence à 35. Il faut prévoir suffisamment de bits pour la quantité ajoutée à 35. Si la valeur 43 est exclue, on pourrait se contenter de 3 bits mais comme elle est incluse nous devons utiliser 4 bits pour la partie entière (le décalage peut prendre les valeurs [0, 15]). Cela nous laisse quand même 4 bits pour la partie fractionnaire, ce qui est beaucoup mieux que le cas précédent. L'erreur absolue maximum est donnée par le poids faible de la partie fractionnaire, c'est-à-dire $2^{-4} = 0,0625$ degré. On peut afficher une valeur pour chaque dixième de degré ce qui est suffisant.

7 Représentation des nombres flottants, précision absolue et relative

7.1 cas simple :

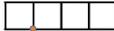
Avec 2 bits d'exposant, on aura $2^2 = 4$ intervalles successifs dans lesquels les nombres représentés auront le même écart entre eux. Il y a $2^3 = 8$ nombres par intervalle. On utilise ici seulement la formule normalisée suivante : $2^{\text{exposant}} \times 1,\text{mantisse}$.

1. Le min est donné par : $2^0 \times 1,0$ c'est-à-dire 1. Pour le max :

Exposant = 11 c'est-à-dire 3,
Mantisse = 111 c'est-à-dire 0.875,

Le max est donné par : $2^3 \times 1,875 = 8 \times 1,875 = 15$.

2. Autres valeurs représentés, groupées sur une ligne pour chaque valeur de l'exposant

1	1,125	1,25	1,375	1,5	1,625	1,75	1,875	
2	2,25	2,5	2,75	3	3,25	3,5	3,75	
4	4,5	5	5,5	6	6,5	7	7,5	
8	9	10	11	12	13	14	15	

3. L'erreur absolue maximum n'est pas constante : c'est l'écart entre 2 nombres représentés successifs. On a successivement une erreur absolue maximum de 0,125, puis 0,25, puis 0,5 puis 1. Cet écart double quand on passe d'un intervalle associé à une puissance de 2 à la puissance de 2 suivante.

4. représentation de certaines valeurs x avec cette représentation:

- $x=1,12$ n'est pas exactement représentée ; elle est approchée par troncation par $x' = 1$.
 - L'erreur absolue sur x est donc 0,12
 - L'erreur relative sur x est $0,12/1,12 =$ environ 10,7%
- $x=1,375$ est exactement représentée ; erreurs absolue et relative nulles
- $x=8,75$ n'est pas exactement représentée ; elle est approchée par 8
 - L'erreur absolue sur x est donc 0,75
 - L'erreur relative sur x est $0,75/8,75 =$ environ 8,6%
- $x=12,75$ n'est pas exactement représentée ; elle est approchée par 12
 - L'erreur absolue sur x est donc 0,75
 - L'erreur relative sur x est $0,75/12,75 =$ environ 5,9%

- l'erreur relative la plus importante se trouve *au début de chaque intervalle associé à une valeur de la puissance de 2* car l'erreur absolue est constante sur tout l'intervalle, et donc on obtient une plus grande valeur d'erreur relative quand on la divise par une valeur x du début d'intervalle.
- la précision est donnée par le poids faible de la mantisse = $2^{-3} = 0.125$ c'est-à-dire 12.5%.

7.2 cas orienté « programmation » de système embarqué avec la virgule flottante en simple précision:

Le cours de programmation vous recommande à juste titre d'utiliser le type « *virgule flottante double précision* », noté **double** en C++, pour déclarer vos variable et effectuer tous vos calculs car il exploite 64 bits pour représenter les nombres réels. De ce point de vue, le cours de C++ a bien raison : travaillez toujours avec le type double pour les nombres réels.

Cela dit, les systèmes embarqués n'ont pas toujours un processeur 64bits à disposition, c'est pourquoi cet exercice a examiné de plus près le format « *virgule flottante simple précision* » qui exploite seulement 32 bits.

- l'exposant vaut 127. La puissance de la base 2 vaut $127-127=0$. Le nombre représenté est $2^0 \times 1,0 = 1$.
- l'exposant vaut 128. La puissance de la base 2 vaut $128-127=1$. Le nombre représenté est $2^1 \times 1,0 = 2$.
- l'exposant vaut 128. La puissance de la base 2 vaut $128-127=1$. Le nombre représenté est $2^1 \times 1,1_2 = 2 \times 1,5 = 3$
- il faut exprimer le motif binaire à partir du nombre en hexadécimal : 11000000010000000000000000000000

On en extrait le bit de signe qui vaut 1 ; ce nombre est donc un nombre négatif

Le motif binaire de « exposant » est 10000000 qui vaut 128. La puissance de la base 2 vaut $128-127=1$.

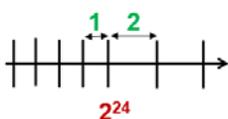
La mantisse vaut 1000000000000000000000 ; le nombre représenté est $- 2^1 \times 1,1_2 = - 2 \times 1,5 = -3$

- La plus grande valeur de « exposant » est un motif binaire avec des 1 partout, c'est-à-dire 11111111 = 255. La puissance de la base 2 vaut $255-127 = 128$. Le plus grand nombre possible est proche de $2^{128} =$ environ $3,4 \cdot 10^{38}$
- le plus petit nombre strictement supérieur à zéro exploite la forme dénormalisée qui permet d'inclure zéro dans l'intervalle des valeurs représentée. C'est la forme exploitée pour la plus petite valeur de « exposant » lorsqu'il n'y a que des 0. La puissance de la base 2 dans la forme dénormalisée est : $(0 - 127) + 1 = 2^{-126}$.

Mais ça n'est pas tout car on exploite aussi la mantisse dans la forme dénormalisée comme ceci : **0,mantisse**
De ce fait, le plus petit nombre représentable strictement supérieur à zéro est celui obtenu avec des zéros partout dans la mantisse sauf pour le poids faible qui vaut 2^{-23} .

Bilan : le plus petit nombre est le produit des deux éléments : $2^{-126} \times 2^{-23} = 2^{-149} =$ env. $1,4 \times 10^{-45}$

- la précision est le poids faible de la mantisse, c'est à dire $2^{-23} =$ env $1,2 \times 10^{-7}$ ce qui garantit seulement que les nombres décimaux avec jusqu'à 6 chiffres significatifs sont représentés ; à partir de 7^{ième} chiffre significatif il y a un risque de plus en plus important d'approximation sur le(s) chiffre(s) de poids faibles.
- sachant que la précision vaut 2^{-23} et que nous cherchons X_1 pour lequel l'erreur absolue δx vaut 1, nous avons l'équation suivante à résoudre : $2^{-23} = 1 / X_1$, donc $X_1 = 2^{23}$
- sachant que la précision vaut 2^{-23} et que nous cherchons X_2 pour lequel l'erreur absolue δx vaut 2, nous avons l'équation suivante à résoudre : $2^{-23} = 2^1 / X_2$, donc $X_2 = 2^{24}$



- la valeur $2^{24} + 1$ n'a pas de représentation, on obtient 2^{24} par troncation
- c'est la puissance de 2 pour laquelle l'erreur absolue maximum vaut 16, c'est-à-dire $2^4 \times 2^{23} = 2^{27}$