Architecture des ordinateurs (E. Lazard)
Examen du 27 janvier 2012

(durée 2 heures) – CORRECTION

I. Nombres flottants

On considère une représentation simplifiée des réels en virgule flottante.

Un nombre réel X est représenté par 10 bits $\boxed{\text{s eeee mmmmm}}$ où $X = (-1)^s * 1{,}m * 2^{e-7}$ avec un exposant sur 4 bits ($0 < e \leq 15$, un exposant égal à 0 désigne le nombre 0 quelle que soit la pseudo-mantisse) et une pseudo-mantisse sur 5 bits (représentant les puissances négatives de 2 ; elles ont pour valeur $2^{-1} = 0{,}5$, $2^{-2} = 0{,}25$, $2^{-3} = 0{,}125$, $2^{-4} = 0{,}0625$ et $2^{-5} = 0{,}03125$).

Les calculs se font sur tous les chiffres significatifs et les arrondis s'effectuent ensuite inférieurement.

1. Représenter 31 et 1,5 en virgule flottante.
2. Pour additionner deux nombres en virgule flottante, il faut décaler la mantisse d'un des deux nombres pour égaliser les exposants, additionner les mantisses (sans oublier le 1 débutant la mantisse), puis renormaliser le nombre en arrondissant éventuellement la pseudo-mantisse.
   Ainsi, on a $7 = 1{,}11000_2 \times 2^2$ et $1{,}25 = 1{,}01000_2 \times 2^0 = 0{,}0101000_2 \times 2^2$ dont l'addition donne $10{,}0001_2 \times 2^2$ qui se normalise en $1{,}00001_2 \times 2^3 = 8{,}25$.
   Expliciter l'addition flottante de 31 et 1,5.
3. On écrit le code suivant :

   Listing 1. ***Calculs***

   ```
   float f = 62.0;
   for (i=0; i<5; i++)
       f = f++;
   ```

   Quelle valeur aura la variable `f` après l'exécution ? Et quelle aurait été sa valeur si on avait simplement écrit :

   ```
   float f = 62.0 + 5;
   ```

Corrigé :

1. $31 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 1{,}11110_2 \times 2^4 = \boxed{0\,1011\,11110}$
   $1{,}5 = 1{,}5 \times 2^0 = \boxed{0\,0111\,10000}$
2. Pour additionner 31 et 1,5, il faut décaler la mantisse de ce dernier de quatre bits vers la droite, ce qui donne $1{,}5 = 1{,}10000_2 \times 2^0 = 0{,}00011_2 \times 2^4$. L'addition donne un résultat égal à $10{,}00001_2 \times 2^4$ que l'on renormalise en $1{,}000001_2 \times 2^5$ et le dernier bit de la mantisse saute à cause de l'arrondi. Le résultat obtenu est alors $1{,}00000_2 \times 2^5 = 32$.
3. Dans le premier cas, on doit additionner un à `f` à chaque passage dans la boucle. 62 s'écrit $1{,}11110_2 \times 2^5$ et 1 s'écrit $1{,}00000 \times 2^0 = 0{,}00001 \times 2^5$. L'addition est correcte la première fois ($1{,}11111_2 \times 2^5$) puis la deuxième ($10{,}0000_2 \times 2^5 = 1{,}00000_2 \times 2^6$) mais ensuite l'arrondi annule l'incrémentation. `f` est donc bloquée à 64.
   $5 = 1{,}25 \times 2^2 = 1{,}01000_2 \times 2^2 = 0{,}00101_2 \times 2^5$ et l'addition avec 62 donne $10{,}00011_2 \times 2^5$ que l'on renormalise en $1{,}000011_2 \times 2^6$ et le dernier bit de la mantisse saute à cause de l'arrondi. Le résultat obtenu est alors $1{,}00001_2 \times 2^6 = 66$.

II. Circuits logiques

Soit une machine qui travaille en entrée sur des nombres binaires signés de 3 bits. La valeur signée d'un mot $N = a_2a_1a_0$ est égale à $a_0 + 2a_1 - 4a_2$ (c'est la représentation classique en complément à 2).

On désire construire un circuit qui donne en sortie le quotient et le reste de la division de l'entrée N par 3 suivant la règle $N = 3q + r$ avec $0 \leq r < 3$.

1. Quel est l'intervalle de valeurs possibles pour l'entrée N ? Quel est l'intervalle de valeurs obtenues pour le quotient q et combien de bits signés sont-ils nécessaires pour exprimer le quotient ? Quel est l'intervalle de valeurs obtenues pour le reste r et combien de bits non-signés sont-ils nécessaires pour exprimer le reste ?
2. Donner la table de vérité du circuit.
3. Donner les expressions logiques des bits de sortie (ceux du quotient et ceux du reste) en fonction des 3 bits d'entrée en simplifiant au maximum les expressions.

Corrigé :

1. Sur 3 bits signés, on exprime les valeurs de -4 à 3. Le quotient va donc de -2 à 1 (nécessitant 2 bits signés) et le reste de 0 à 2 (nécessitant 2 bits non-signés).
2.

a_2	a_1	a_0	q_1	q_0	r_1	r_0
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	1	0
0	1	1	0	1	0	0
1	0	0	1	0	1	0
1	0	1	1	1	0	0
1	1	0	1	1	0	1
1	1	1	1	1	1	0

3. On voit donc que l'on a :

$q_1 = a_2$

$q_0 = \overline{a_2}a_1a_0 + a_2\overline{a_1}a_0 + a_2a_1\overline{a_0} + a_2a_1a_0 = a_1a_0 + a_2(a_1 \oplus a_0)$

$r_1 = \overline{a_2}a_1\overline{a_0} + a_2\overline{a_1}\,\overline{a_0} + a_2a_1a_0 = (a_2 \oplus a_1)\overline{a_0} + a_2a_1a_0$

$r_0 = \overline{a_2}\,\overline{a_1}a_0 + a_2a_1\overline{a_0}$

III. Assembleur *(Les deux questions sont indépendantes)*

1. Une chaîne de caractères est stockée en mémoire. Chaque caractère (une des 26 lettres majuscules) est stocké sur un octet et l'octet qui suit le dernier caractère de la chaîne est nul. L'adresse du premier caractère de la chaîne se trouve dans le registre `r0`.

   On souhaite recopier cette chaîne en supprimant les répétitions successives d'un caractère. Ainsi, la chaîne `ABBCAAFGGHGGG` devra être recopiée en `ABCAFGHG` (on ne supprime pas toutes les répétitions de caractères mais on ne garde qu'un caractère pour chaque bloc de répétitions). Cette nouvelle chaîne devra être recopiée à partir de l'adresse contenue dans le registre `r1`. Écrire la procédure assembleur correspondante.
2. Une chaîne de caractères est stockée en mémoire. Chaque caractère (une des 26 lettres majuscules) est stocké sur un octet et l'octet qui suit le dernier caractère de la chaîne est nul. L'adresse du premier caractère de la chaîne se trouve dans le registre `r0`.

   On souhaite recopier cette chaîne en dupliquant chaque caractère autant de fois que son rang dans l'alphabet (`A` est de rang 1, `B` de rang 2, etc.). Ainsi, la chaîne `ADCAB` devra être recopiée

en ADDDDCCCABB. Cette nouvelle chaîne devra être recopiée à partir de l'adresse contenue dans le registre r1. Écrire la procédure assembleur correspondante.

Corrigé :

1.

Listing 2. ***Enlever les répétitions***

```
        MVI     r3,#0           ; mémoriser le dernier caractère
loop:   LDB     r2,(r0)         ; charger le prochain
        ADD     r0,r0,#1        ; décaler le pointeur
        JZ      r2,fin          ; est-ce fini ?
        SUB     r31,r2,r3       ; identique au précédent caractère ?
        JZ      r31,loop        ; si oui on passe au suivant
        STB     (r1),r2         ; sinon on le stocke
        MOV     r3,r2           ; et on le mémorise comme caractère courant
        ADD     r1,r1,#1        ; on avance le pointeur
        JMP     loop            ; au suivant !
fin:    STB     (r1),r2         ; le zéro final
```

2.

Listing 3. ***Répéter les caractères***

```
loop:   LDB     r2,(r0)         ; charger le prochain caractère
        JZ      r2,fin          ; fini ?
        SUB     r3,r2,#64       ; quel est son rang ?
repet:  STB     (r1),r2         ; on le stocke autant de fois
        ADD     r1,r1,#1        ;
        SUB     r3,r3,#1        ; que son rang
        JNZ     r3,repet        ; fin de la répétition ?
        ADD     r0,r0,#1        ; avancer le pointeur
        JMP     loop            ; caractère suivant
fin:    STB     (r1),r2         ; le zéro final
```

IV. Mémoire cache

Un programme se compose d'une boucle de 20 instructions à exécuter 3 fois ; les instructions se trouvant, dans l'ordre, aux adresses mémoire 1 à 4, 21 à 24, 1 à 4, 25 à 28 et 1 à 4. Ce programme doit tourner sur une machine possédant un cache d'une taille de 18 instructions. Le temps de cycle de la mémoire principale est M et le temps de cycle du cache est C. Le cache est associatif (un bloc mémoire peut venir dans n'importe quel bloc du cache) et la stratégie de remplacement utilisée est LRU (on remplace le bloc le moins récemment utilisé).

1. Le cache possède 6 blocs de 3 instructions : les blocs que l'on peut transférer sont 1-3, 4-6, ..., 19-21, 22-24, 25-27, 28-30... Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ? Le cache est vide au départ.
2. Le cache possède 3 blocs de 6 instructions : les blocs que l'on peut transférer sont 1-6, ..., 19-24, 25-30... Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ? Le cache est vide au départ.
3. Le cache possède 2 blocs de 9 instructions : les blocs que l'on peut transférer sont 1-9, ..., 19-27, 28-36... Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ? Le cache est vide au départ.

4. Le cache possède 1 blocs de 18 instructions : les blocs que l'on peut transférer sont 1-18, 19-36... Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ? Le cache est vide au départ.
5. Qu'en concluez-vous sur l'efficacité du cache ?

Rappel : Lorsque l'on va chercher un mot en mémoire, pendant M on ramène le mot pour le processeur et tout le bloc correspondant dans le cache.

Corrigé :

1.

$1 \to 3$	$M + 2C$	bloc 1	$1 \to 3$	$3C$	bloc 1	$1 \to 3$	$3C$	bloc 1
4	M	bloc 2	4	C	bloc 2	4	C	bloc 2
21	M	bloc 3	21	C	bloc 3	21	C	bloc 3
$22 \to 24$	$M + 2C$	bloc 4	$22 \to 24$	$3C$	bloc 4	$22 \to 24$	$3C$	bloc 4
$1 \to 3$	$3C$	bloc 1	$1 \to 3$	$3C$	bloc 1	$1 \to 3$	$3C$	bloc 1
4	C	bloc 2	4	C	bloc 2	4	C	bloc 2
$25 \to 27$	$M + 2C$	bloc 5	$25 \to 27$	$3C$	bloc 5	$25 \to 27$	$3C$	bloc 5
28	M	bloc 6	28	C	bloc 6	28	C	bloc 6
$1 \to 3$	$3C$	bloc 1	$1 \to 3$	$3C$	bloc 1	$1 \to 3$	$3C$	bloc 1
4	C	bloc 2	4	C	bloc 2	4	C	bloc 2

Soit un total de $6M + 54C$.

2.

$1 \to 4$	$M + 3C$	bloc 1	$1 \to 4$	$4C$	bloc 1	$1 \to 4$	$4C$	bloc 1
$21 \to 24$	$M + 3C$	bloc 2	$21 \to 24$	$4C$	bloc 2	$21 \to 24$	$4C$	bloc 2
$1 \to 4$	$4C$	bloc 1	$1 \to 4$	$4C$	bloc 1	$1 \to 4$	$4C$	bloc 1
$25 \to 28$	$M + 3C$	bloc 3	$25 \to 28$	$4C$	bloc 3	$25 \to 28$	$4C$	bloc 3
$1 \to 4$	$4C$	bloc 1	$1 \to 4$	$4C$	bloc 1	$1 \to 4$	$4C$	bloc 1

Soit un total de $3M + 57C$.

3.

$1 \to 4$	$M + 3C$	bloc 1	$1 \to 4$	$4C$	bloc 2	$1 \to 4$	$4C$	bloc 1
$21 \to 24$	$M + 3C$	bloc 2	$21 \to 24$	$M + 3C$	bloc 1	$21 \to 24$	$M + 3C$	bloc 2
$1 \to 4$	$4C$	bloc 1	$1 \to 4$	$4C$	bloc 2	$1 \to 4$	$4C$	bloc 1
$25 \to 27$	$3C$	bloc 2	$25 \to 27$	$3C$	bloc 1	$25 \to 27$	$3C$	bloc 2
28	M	bloc 1	28	M	bloc 2	28	M	bloc 1
$1 \to 4$	$M + 3C$	bloc 2	$1 \to 4$	$M + 3C$	bloc 1	$1 \to 4$	$M + 3C$	bloc 2

Soit un total de $10M + 50C$.

4.

$1 \to 4$	$M + 3C$	bloc 1	$1 \to 4$	$4C$	bloc 1	$1 \to 4$	$4C$	bloc 1
$21 \to 24$	$M + 3C$	bloc 1	$21 \to 24$	$M + 3C$	bloc 1	$21 \to 24$	$M + 3C$	bloc 1
$1 \to 4$	$M + 3C$	bloc 1	$1 \to 4$	$M + 3C$	bloc 1	$1 \to 4$	$M + 3C$	bloc 1
$25 \to 28$	$M + 3C$	bloc 1	$25 \to 28$	$M + 3C$	bloc 1	$25 \to 28$	$M + 3C$	bloc 1
$1 \to 4$	$M + 3C$	bloc 1	$1 \to 4$	$M + 3C$	bloc 1	$1 \to 4$	$M + 3C$	bloc 1

Soit un total de $13M + 47C$.

5. L'efficacité du cache n'est pas proportionnelle à la taille de ses blocs. De gros blocs permettent de mémoriser plus d'information à chaque accès mémoire mais de petits blocs permettent de garder des parties du programme qui sont souvent réutilisées.