Architecture des ordinateurs (E. Lazard)
Examen du 18 janvier 2016

(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 ($0 < e \leqslant 15$, 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 lorsque cela est indiqué.

1. Représenter 1,25 ; 5 et 6,25 en virgule flottante.
2. Calculer les résultats exacts et arrondis des opérations de multiplication flottante $5 \times 6{,}25$ et $5 \times 1{,}25$.
3. On écrit le code suivant :

Listing 1. Calculs

```
float f = 6.25;
float g = 1.25;
float x = 5*f - 5*g;
```

   (a) Quelle valeur obtient-on pour `x` si on suppose que tous les calculs sont exacts (sans arrondi) ?
   (b) On suppose qu'après chacune des trois opérations, la valeur obtenue est remise en mémoire *et donc arrondie inférieurement à ce moment-là.* Quelle valeur obtient-on pour `x` ?
   (c) Le compilateur a optimisé la troisième ligne de code qui devient `float x = 5*(f-g);`. Il n'y a plus maintenant que deux arrondis, un après la soustraction et un second après la multiplication. Quelle valeur obtient-on pour `x` ?
   (d) Quelle conclusion en tirez-vous ?

Corrigé :

1. $1{,}25 = 1{,}25 \times 2^0 = \boxed{0\,0111\,01000}$
   $5 = 1{,}25 \times 2^2 = \boxed{0\,1001\,01000}$
   $6{,}25 = 1{,}5625 \times 2^2 = \boxed{0\,1001\,10010}$
2. $5 \times 6{,}25$ donne un résultat exact de 31,25. La multiplication flottante donne $1{,}01_2 \times 2^2 \times 1{,}1001_2 \times 2^2 = 1{,}111101_2 \times 2^4$ qui est arrondi à $1{,}11110_2 \times 2^4 = 31$.
   $5 \times 1{,}25$ donne un résultat exact de 6,25. La multiplication flottante donne $1{,}01_2 \times 2^2 \times 1{,}01_2 \times 2^0 = 1{,}1001 \times 2^2 = 6{,}25$ et aucun arrondi n'est à faire.
3. (a) Si les calculs étaient parfaits, `x` vaudrait $31{,}25 - 6{,}25 = 25$.
   (b) `5f` est arrondi à $1{,}11110_2 \times 2^4$ et `5g` donne $1{,}1001 \times 2^2$. On doit donc calculer $1{,}11110_2 \times 2^4 - 1{,}1001 \times 2^2 = 1{,}11110_2 \times 2^4 - 0{,}011001_2 \times 2^4 = 1{,}100011 \times 2^4$ qui vaut 24,75 et est arrondi à $1{,}10001 \times 2^4 = 24{,}5$.

(c) `f-g` se calcule par $1{,}1001_2 \times 2^2 - 1{,}01_2 \times 2^0 = 1{,}1001_2 \times 2^2 - 0{,}0101_2 \times 2^2 = 1{,}0100_2 \times 2^2 = 5$, sans arrondi nécessaire. Et la multiplication par 5 donne : $1{,}01_2 \times 2^2 \times 1{,}01_2 \times 2^2 = 1{,}1001_2 \times 2^4 = 25$ qui est bien le résultat exact.

(d) Une simple optimisation du compilateur peut changer le résultat final. C'est un des soucis de la norme de calcul en virgule flottante : ceux-ci ne sont pas forcément toujours reproductibles.

II. Circuits logiques

On cherche à faire une UAL simplifiée comme suit : on veut un circuit à deux entrées a et b, une ligne de commande F et deux sorties s_0 et s_1 tel que :

- si $F = 0$, le circuit se comporte comme un demi-additionneur, la sortie s_0 représentant la somme des deux bits et s_1 la retenue ;
- si $F = 1$, le circuit se comporte comme une unité logique, la sortie s_0 représentant alors le OU des deux entrées et la sortie s_1 le ET des deux entrées.

1. Construire les deux tables de vérité (pour $F = 0$ et $F = 1$).
2. Exprimer s_0 et s_1 en fonction de a, b et F.
3. En remarquant que le OU peut se décomposer en OU, XOR et ET, simplifier les sorties et représenter le circuit à l'aide de 4 portes logiques.

Corrigé :

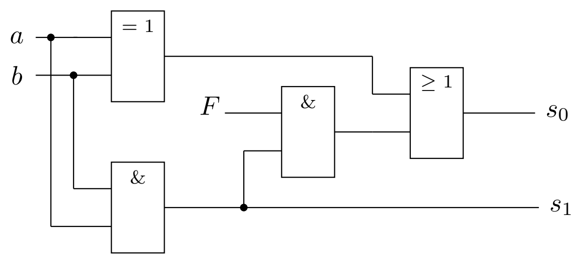
1. Pour $F = 0$

a	b	s_0	s_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

pour $F = 1$

a	b	s_0	s_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

2. $s_0 = (a \oplus b)\overline{F} + (a + b)F$
   $s_1 = ab$
3. On remarque que $a + b = (a \oplus b) + ab$.
   Cela nous donne $s_0 = (a \oplus b) + abF$ et donc le circuit suivant :



III. Assembleur

1. Une chaîne de caractères est stockée en mémoire. Chaque caractère est stocké sur un octet et l'octet qui suit le dernier caractère de la chaîne est nul. L'adresse du premier élément de la chaîne se trouve dans le registre `r0` et on pourra supposer que la chaîne n'est pas vide.
   La chaîne est composée de caractères minuscules (`'a'` à `'z'`) chacun suivi d'un caractère numérique (donc entre `'0'` et `'9'`). Ainsi par exemple on peut avoir la chaine `"a1b3w0z6"`. On souhaite recopier la chaîne en remplaçant chaque couple caractère-chiffre par la répétition du caractère autant de fois que

le chiffre indiqué : "a1b3w0z6" va générer "abbbzzzzzz". Cette nouvelle chaîne sera stockée à partir de l'adresse contenue dans le registre r1. Écrire la procédure assembleur correspondante ; on ne vous demande pas de conserver la valeur ni de r0 ni de r1 à la fin. *(On pourra utiliser #'0' pour avoir le code ASCII du caractère '0'.)*

2. *(Plus difficile)* On souhaite maintenant effectuer l'opération inverse. r0 pointe sur une chaîne composée de lettres minuscules dans l'ordre alphabétique et présents un certain nombre de fois (9 ou moins), par exemple "aabccchhhhhwxxxxxxxxzz". On souhaite recopier cette chaîne en remplaçant chaque bloc formé d'une même lettre par cette lettre et son nombre d'occurrences. Pour l'exemple précédent, on devrait obtenir "a2b1c3h5w1x8z2". Cette nouvelle chaîne sera stockée à partir de l'adresse contenue dans le registre r1. Écrire la procédure assembleur correspondante ; on ne vous demande pas de conserver la valeur ni de r0 ni de r1 à la fin et on pourra supposer que la chaîne initiale n'est pas vide.

Corrigé :

1.

Listing 2. Recopier les occurrences

```
loop:   LDB    r10,(r0)       ; charger un caractère
        JZ     r10, fin       ; fin de la chaîne ?
        ADD    r0,r0,#1       ; avancer pointeur
        LDB    r11,(r0)       ; et récupérer le nombre d'occurrences
        ADD    r0,r0,#1
        SUB    r11,r11,#'0'   ; retomber sur un chiffre et pas un caractère num.
copie:  JZ     r11,loop       ; si on a fini de recopier, passer au caractère suivant
        STB    (r1),r10       ; sinon on le recopie encore une fois
        ADD    r1,r1,#1       ; on décale le pointeur de recopie
        SUB    r11,r11,#1     ; et on décrémente le compteur
        JMP    copie
fin:    STB    (r1),r10       ; on termine la chaîne avec un 0 final.
```

2.

Le premier algorithme auquel on peut penser est le suivant : on compte les occurrences d'un caractère puis lorsqu'il change, on stocke l'ancien caractère suivi de son compte.

Listing 3. Compter les occurrences

```
        MVI    r11,#0         ; sauvegarde du caractère précédent
        MVI    r12,#0         ; compteur
loop:   LDB    r20,(r0)       ; on charge le caractère suivant
        ADD    r0,r0,#1
        SUB    r31,r20,r11    ; est-il différent du précédent
        JNZ    r31,diff       ; si oui, on traite
        ADD    r12,r12,#1     ; sinon, on incrémente le compteur d'occurrences
        JMP    loop
diff:   JZ     r12,prem       ; Est-ce le 1er car. de la chaîne (compteur préc. nul) ?
        ADD    r12,r12,#'0'   ; sinon, on stocke le compteur précédent
        STB    (r1),r12
        ADD    r1,r1,#1
prem:   STB    (r1),r20       ; dans tous les cas on stocke le nouveau caractère
        ADD    r1,r1,#1
        MOV    r11,r20        ; on le sauvegarde
        MVI    r12,#1         ; et le compteur est initialisé à 1
        JNZ    r20,loop
fin:    STB    (r1),r20       ; on met le 0 final.
```

Mais en fait il y a plus simple : dès qu'on tombe sur un nouveau caractère, on le stocke dans la nouvelle chaîne, puis on compte ses occurrences ; lorsque le caractère change, on stocke son compte et on reboucle.

Listing 4. Compter les occurrences 2

```
debut:  MVI r4,#'1'           ; initialiser le compteur
        LDB r30,(r0)          ; charger le caractère
        JZ  r30,fin           ; est-ce terminé ?
        STB (r1),r30          ; on recopie ce caractère
ici:    ADD r0,r0,#1          ; passe au caractère suivant
        LDB r31,(r0)
        SUB r5,r30,r31        ; est-il différent ?
        JNZ r5,suite
        ADD r4,r4,#1          ; si non, on incrémente le compteur d'occurrences
        JMP ici
suite:  ADD r1,r1,#1          ; on commence un nouveau caractère
        STB (r1),r4           ; on recopie le décompte précédent
        ADD r1,r1,#1
        JMP debut             ; et on recommence avec ce nouveau caractère
fin:    STB (r1),r30          ; on met le 0 final.
```

IV. Mémoire cache

Un programme se compose d'une boucle de 20 instructions à exécuter 4 fois ; les instructions se trouvant, dans l'ordre, aux adresses mémoire 13 à 16, 1 à 4, 17 à 20, 1 à 4, 13 à 16. Ce programme doit tourner sur une machine possédant un cache d'une taille de 8 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 4 blocs de 2 instructions : les blocs que l'on peut transférer sont 1-2, 3-4, ..., 13-14, 15-16, 17-18, 19-20... 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 2 blocs de 4 instructions : les blocs que l'on peut transférer sont 1-4, 5-8, ..., 13-16, 17-20... 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. Refaire la première question en supposant que le cache est à correspondance directe, formé de 4 blocs de 2 instructions. On rappelle que cela veut dire que les blocs 1-2, 9-10, 17-18... vont dans l'emplacement 1 du cache, que les blocs 3-4, 11-12, 19-20... vont dans le deuxième, etc. Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ?

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.

<table>
<tr><td>$13 \to 14$</td><td>$M + C$</td><td>bloc 1</td><td>$13 \to 14$</td><td>$2C$</td><td>bloc 1</td><td rowspan="10">Les deux dernières itérations sont identiques</td></tr>
<tr><td>$15 \to 16$</td><td>$M + C$</td><td>bloc 2</td><td>$15 \to 16$</td><td>$2C$</td><td>bloc 2</td></tr>
<tr><td>$1 \to 2$</td><td>$M + C$</td><td>bloc 3</td><td>$1 \to 2$</td><td>$2C$</td><td>bloc 3</td></tr>
<tr><td>$3 \to 4$</td><td>$M + C$</td><td>bloc 4</td><td>$3 \to 4$</td><td>$2C$</td><td>bloc 4</td></tr>
<tr><td>$17 \to 18$</td><td>$M + C$</td><td>bloc 1</td><td>$17 \to 18$</td><td>$M + C$</td><td>bloc 1</td></tr>
<tr><td>$19 \to 20$</td><td>$M + C$</td><td>bloc 2</td><td>$19 \to 20$</td><td>$M + C$</td><td>bloc 2</td></tr>
<tr><td>$1 \to 2$</td><td>$2C$</td><td>bloc 3</td><td>$1 \to 2$</td><td>$2C$</td><td>bloc 3</td></tr>
<tr><td>$3 \to 4$</td><td>$2C$</td><td>bloc 4</td><td>$3 \to 4$</td><td>$2C$</td><td>bloc 4</td></tr>
<tr><td>$13 \to 14$</td><td>$M + C$</td><td>bloc 1</td><td>$13 \to 14$</td><td>$M + C$</td><td>bloc 1</td></tr>
<tr><td>$15 \to 16$</td><td>$M + C$</td><td>bloc 2</td><td>$15 \to 16$</td><td>$M + C$</td><td>bloc 2</td></tr>
</table>

Soit un total de $20M + 60C$.

2.

<table>
<tr><td>$13 \to 16$</td><td>$M + 3C$</td><td>bloc 1</td><td>$13 \to 16$</td><td>$4C$</td><td>bloc 1</td><td rowspan="5">Les deux dernières itérations sont identiques</td></tr>
<tr><td>$1 \to 4$</td><td>$M + 3C$</td><td>bloc 2</td><td>$1 \to 4$</td><td>$4C$</td><td>bloc 2</td></tr>
<tr><td>$17 \to 20$</td><td>$M + 3C$</td><td>bloc 1</td><td>$17 \to 20$</td><td>$M + 3C$</td><td>bloc 1</td></tr>
<tr><td>$1 \to 4$</td><td>$4C$</td><td>bloc 2</td><td>$1 \to 4$</td><td>$4C$</td><td>bloc 2</td></tr>
<tr><td>$13 \to 16$</td><td>$M + 3C$</td><td>bloc 1</td><td>$13 \to 16$</td><td>$M + 3C$</td><td>bloc 1</td></tr>
</table>

Soit un total de $10M + 70C$.

3.

<table>
<tr><td>$13 \to 14$</td><td>$M + C$</td><td>bloc 3</td><td>$13 \to 14$</td><td>$2C$</td><td>bloc 3</td><td rowspan="10">Les deux dernières itérations sont identiques</td></tr>
<tr><td>$15 \to 16$</td><td>$M + C$</td><td>bloc 4</td><td>$15 \to 16$</td><td>$2C$</td><td>bloc 4</td></tr>
<tr><td>$1 \to 2$</td><td>$M + C$</td><td>bloc 1</td><td>$1 \to 2$</td><td>$2C$</td><td>bloc 1</td></tr>
<tr><td>$3 \to 4$</td><td>$M + C$</td><td>bloc 2</td><td>$3 \to 4$</td><td>$2C$</td><td>bloc 2</td></tr>
<tr><td>$17 \to 18$</td><td>$M + C$</td><td>bloc 1</td><td>$17 \to 18$</td><td>$M + C$</td><td>bloc 1</td></tr>
<tr><td>$19 \to 20$</td><td>$M + C$</td><td>bloc 2</td><td>$19 \to 20$</td><td>$M + C$</td><td>bloc 2</td></tr>
<tr><td>$1 \to 2$</td><td>$M + C$</td><td>bloc 1</td><td>$1 \to 2$</td><td>$M + C$</td><td>bloc 1</td></tr>
<tr><td>$3 \to 4$</td><td>$M + C$</td><td>bloc 2</td><td>$3 \to 4$</td><td>$M + C$</td><td>bloc 2</td></tr>
<tr><td>$13 \to 14$</td><td>$2C$</td><td>bloc 3</td><td>$13 \to 14$</td><td>$2C$</td><td>bloc 3</td></tr>
<tr><td>$15 \to 16$</td><td>$2C$</td><td>bloc 4</td><td>$15 \to 16$</td><td>$2C$</td><td>bloc 4</td></tr>
</table>

Soit un total de $20M + 60C$.