UNIVERSITÉ DE PARIS-DAUPHINE L2 MI2E

Architecture des ordinateurs (E. Lazard)
Examen du 18 janvier 2019

(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 l'arrondi s'effectue ensuite inférieurement après chaque opération.

1. Représenter 7 et 17 en virgule flottante.
2. Calculer la multiplication virgule flottante de 7 et 17. Quelle est la valeur obtenue ?
3. 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.
   Comparer les valeurs obtenues pour chacun des deux calculs suivants effectués en virgule flottante : $(17 - 1) \times 7$ et $17 \times 7 - 7$.

Corrigé :

1. $7 = 1{,}75 \times 2^2 = \boxed{0\,1001\,11000}$
   $17 = 1{,}0625 \times 2^4 = \boxed{0\,1011\,00010}$
2. La multiplication des deux mantisses $1{,}11000_2$ et $1{,}00010_2$ donne $1{,}11011\not{1}_2$ comme résultat et le dernier bit de la mantisse saute à cause de l'arrondi.
   Le résultat obtenu est alors $1{,}11011_2 \times 2^{2+4} = 2^6 + 2^5 + 2^4 + 2^2 + 2^1 = 118$ (pour une valeur exacte de 119).
3. $17 - 1 = 16$ s'écrit exactement $1{,}00000_2 \times 2^4$ qui multiplié par $7 = 1{,}11000_2 \times 2^2$, donne $1{,}11000_2 \times 2^6 = 112$. En revanche, 17×7 s'écrit $1{,}11011_2 \times 2^6$ d'où on soustrait $7 = 1{,}11000_2 \times 2^2 = 0{,}000111_2 \times 2^6$. Cela donne $1{,}10111\not{1}_2 \times 2^6$ qui vaut $2^6 + 2^5 + 2^3 + 2^2 + 2^1 = 110$. Notons que si nous avions arrondi supérieurement le dernier résultat, nous serions retombés sur la valeur exacte.

II. Circuits logiques

Soit une machine travaillant sur des nombres binaires signés de 3 bits. La valeur signée d'un mot $A = a_2a_1a_0$ 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 un nombre binaire $B = b_2b_1b_0$ représentant le même nombre A mais cette fois écrit en représentation « signe et valeur absolue » (b_2 étant le signe et b_1b_0 la valeur absolue).

1. Toutes les valeurs sont-elles autorisées en entrée ? Autrement dit, le circuit donne-t-il toujours une valeur correcte ?
2. Donner la table de vérité du circuit.
3. Donner les expressions logiques des 3 bits de sortie en fonction des 3 bits d'entrée en **simplifiant au maximum les expressions**.
4. Donner l'expression logique d'un bit supplémentaire en sortie, e *(débordement)*, qui indiquerait qu'une valeur interdite est présente sur les entrées.

Corrigé :

1. La valeur $-4 = 100_2$ ne peut pas se représenter en convention « signe et valeur absolue ».
2.

A	a_2	a_1	a_0	b_2	b_1	b_0
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	0
3	0	1	1	0	1	1
-4	1	0	0	–	–	–
-3	1	0	1	1	1	1
-2	1	1	0	1	1	0
-1	1	1	1	1	0	1

3. On obtient:

$$b_0 = a_0$$

$$b_1 = \overline{a_2}a_1\overline{a_0} + \overline{a_2}a_1a_0 + a_2\overline{a_1}a_0 + a_2a_1\overline{a_0}$$

qui s'écrit aussi :

$$b_1 = \overline{a_2}a_1 + a_2(a_1 \oplus a_0)$$

$$b_2 = a_2$$

car le bit de poids fort représente toujours le signe.

4. Le bit de débordement est mis à 1 pour la seule valeur interdite, -4 : $e = a_2\overline{a_1}\,\overline{a_0}$.

III. Assembleur

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 1. 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 2. Répéter les caractères

```
loop:   LDB     r2,(r0)         ; charger le prochain caractère
        JZ      r2,fin          ; fini ?
        SUB     r3,r2,#'A'-1    ; 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
```

V. Mémoire cache

Un programme se compose d'une boucle de 38 instructions à exécuter 10 fois ; cette boucle se trouve aux adresses mémoire 1 à 38. Ce programme doit tourner sur une machine possédant un cache d'une taille de 32 instructions. Le temps de cycle de la mémoire principale est M et le temps de cycle du cache est C.

1. Le cache est à correspondance directe, formé de 16 blocs de 2 instructions. On rappelle que cela veut dire que les mots mémoire 1 et 2, 33 et 34... vont dans le premier bloc, que les mots 3 et 4, 35 et 36... vont dans le deuxième, etc. Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ?
2. Le cache est maintenant 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). Quel est le temps d'exécution du programme ?
3. Refaire les deux premières questions en prenant un cache composé de 8 blocs de 4 mots.

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. Les trois premiers blocs sont partagés ($1-2$ et $33-34$, $3-4$ et $35-36$, $5-6$ et $37-38$), donc :

$$T = 16(M+C) + 3(M+C) + 9[6(M+C) + 26C] = 73M + 307C$$

2. On ne réutilise jamais un bloc, donc :

$$T = 10[19(M+C)] = 190M + 190C$$

3. (a) Les deux premiers blocs sont partagés ($1 \to 4$ et $33 \to 36$, $5 \to 8$ et $37-38$), donc :

$$T = 9(M+3C) + (M+C) + 9[3(M+3C) + (M+C) + 24C] = 46M + 334C$$

   (b) On ne réutilise jamais un bloc, donc :

$$T = 10[9(M+3C) + (M+C)] = 100M + 280C$$