**UNIVERSITÉ DE PARIS-DAUPHINE** **DE MI2E 2e année**

# Architecture des ordinateurs (E. Lazard)
# Examen du 28 janvier 2010

CORRECTION

(durée 2 heures)

## I. Nombres flottants

On considère une représentation simplifiée des réels positifs en virgule **fixe**.

Un nombre réel $X$ est représenté par 16 bits $\boxed{a_{10}a_9...a_1a_0,a_{-1}a_{-2}...a_{-5}}$ où $X = \sum_{i=-5}^{10} a_i 2^i$ (on a donc une partie entière sur 11 bits et une partie fractionnaire 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 $2^{-5} = 0{,}03125$).

1. Donner la valeur décimale du plus grand et du plus petit nombre non-nul représentable.
2. Quel est le nombre représentable le plus proche de 0,1 ? Quel est son écart $\epsilon$ avec 0,1 ?
3. Votre ordinateur possède une horloge qui incrémente un compteur $N$ tous les $\frac{1}{10}$ de seconde (le compteur est mis à 0 au démarrage). Pour afficher l'heure depuis le démarrage, il calcule $N \times 0{,}1$ mais comme 0,1 se représente avec une légère erreur $\epsilon$, le calcul est erroné. Donner, en fonction de $N$ et $\epsilon$, l'écart entre l'heure affichée et l'heure vraie. L'heure affichée est-elle en avance ou en retard ? Au bout de combien de temps cet écart atteint-il une seconde ?

Corrigé :

1. Le plus grand nombre est $\boxed{1111111111{,}11111}$ soit $2^{11} - 2^{-5} = 2047{,}96875$
   Le plus petit nombre non-nul est $\boxed{00000000000{,}00001}$ soit $2^{-5} = 0{,}03125$
2. 0,1 est encadré par 0,125 et $0{,}0625 + 0{,}03125 = 0{,}09375$. C'est ce dernier nombre qui est le plus proche et l'écart est $\epsilon = 0{,}1 - 0{,}09375 = 0{,}00625$.
3. L'heure vraie est $(N \times 0{,}1)$, l'heure affichée est $N \times (0{,}1 - \epsilon)$. L'heure affichée est donc en retard sur l'heure vraie et cet écart est égal à $N \times \epsilon$ secondes. L'écart atteint une seconde quand $N \times \epsilon > 1$, c'est-à-dire $N > \epsilon^{-1} = 0{,}00625^{-1} = 160$. L'écart atteint donc une seconde au bout de 16 secondes vraies (et l'ordinateur affiche alors 15 secondes).

## II. Circuits logiques

On souhaite construire un décrémenteur sur $n$ bits, c'est-à-dire un circuit à $n$ bits d'entrée $A = a_{n-1} \dots a_0$ représentant un nombre binaire non signé, qui génère $n$ bits de sortie $S = s_{n-1} \dots s_0$ représentant la valeur binaire $A - 1$, ainsi qu'un bit de débordement $X$ valant 1 si la valeur $A - 1$ ne peut pas être représentée sur $n$ bits.

1. On appelle $r_i$ la retenue intermédiaire utilisée pour calculer $s_{i+1} = a_{i+1} - r_i$. Donner la table de vérité de $s_0$ et $r_0$ en fonction de $a_0$. Donner les expressions logiques de $s_0$ et $r_0$.
2. Donner la table de vérité de $s_i$ et $r_i$ en fonction de $a_i$ et $r_{i-1}$. Donner les expressions logiques de $s_i$ et $r_i$.
3. À quel bit est égal le bit de débordement $X$ ?

4. Dessiner le circuit donnant $S = s_{n-1} \dots s_0$ et $X$, en fonction des $a_{n-1} \dots a_0$ (vous pouvez utiliser toutes les portes à deux entrées).
5. La retenue finale se calculant en propageant toutes les retenues intermédiaires, son calcul est très long. Donner une expression des $s_i$ et $r_i$ permettant de les calculer beaucoup plus rapidement (sous réserve de disposer des bonnes portes logiques).

Corrigé :

1.

| $a_0$ | $s_0$ | $r_0$ |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 0 | 0 |

$s_0 = \overline{a_0}$

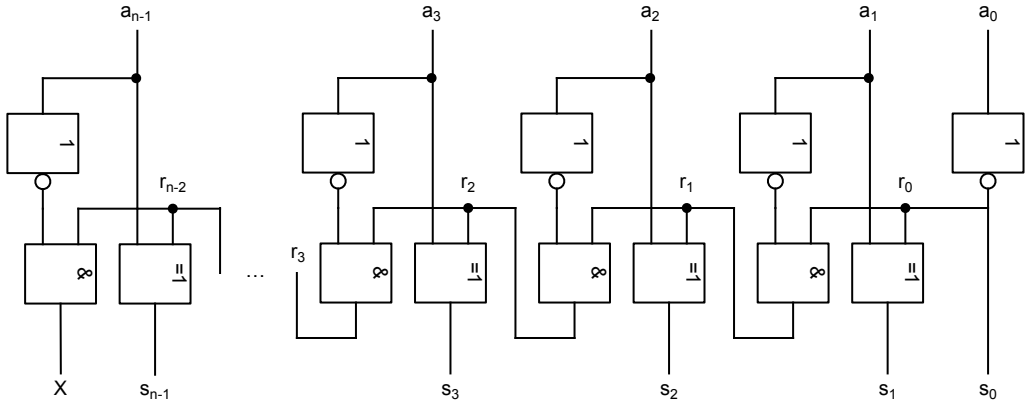
$r_0 = \overline{a_0}$

2.

| $a_i$ | $r_{i-1}$ | $s_i$ | $r_i$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

$s_i = a_i \oplus r_{i-1}$

$r_i = \overline{a_i}.r_{i-1}$

3. $X$ est en fait équivalent à la retenue finale $r_{n-1}$.
4. 

5. Par récurrence, on peut écrire

$$r_i = \overline{a_i}.\overline{a_{i-1}} \dots \overline{a_1}.\overline{a_0} = \prod_{j=0}^{i} \overline{a_j}$$

$$s_i = a_i \oplus (\overline{a_{i-1}}.\overline{a_{i-2}} \dots \overline{a_1}.\overline{a_0}) = a_i \oplus \prod_{j=0}^{i-1} \overline{a_j}$$

## III. Assembleur

1. Une chaîne de caractères est stockée en mémoire. Chaque caractère (une des 26 lettres minuscules) 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`. Les caractères sont dans l'ordre alphabétique et présents un certain nombre de fois (éventuellement 0) : la chaîne est de la forme `"a...ab...bc...cd ... wx...xy...yz...z"`.
   On souhaite déterminer le caractère dont la chaîne correspondante est la plus grande. Ainsi, par exemple, pour la chaîne `"aabcdeeefghiklmxyz"`, c'est le `e` qui apparaît le plus. Écrire une procédure assembleur qui, lorsqu'elle se termine, laisse dans le registre `r1` le caractère le plus fréquent de la chaîne et dans `r2` son nombre d'apparitions.
2. *(plus difficile)* Les caractères ne sont maintenant plus par ordre alphabétique. On a donc une chaîne composée à l'aide des 26 lettres minuscules et on souhaite toujours obtenir le caractère le plus fréquent. On sait qu'aucun caractère n'apparaît plus de 255 fois dans la chaîne. Par ailleurs, on dispose d'un tableau de 26 octets, tous initialisés à 0, sur lequel pointe `r10` (il pointe sur le premier octet du tableau). Écrire une procédure assembleur qui, lorsqu'elle se termine, laisse dans le registre `r1` le caractère le plus fréquent de la chaîne et dans `r2` son nombre d'apparitions.

Corrigé :

1.
```
                MVI   r2,#0             ; sauvegarde meilleur compteur
                MVI   r11,#0            ; sauvegarde précédent caractère
                MVI   r12,#0            ; compteur courant
                LDB   r20,(r0)          ; vérifier si la chaîne est vide
                JZ    r20, fin          ; si oui, c'est fini.
loop:           LDB   r20,(r0)          ; récupérer le caractère
                ADD   r0,r0,#1          ; avancer le pointeur
                SUB   r31,r20,r11       ; est-ce toujours le même?
                JNZ   r31,diff          ; c'est un nouveau caractère
                ADD   r12,r12,#1        ; sinon, incrémenter le compteur
                JMP   loop              ; et passer au caractère suivant
diff:           SUB   r31,r2,r12        ; plus que le meilleur?
                JGE   r31,pasMieux      ; non, pas mieux
                MOV   r1,r11            ; sinon, on sauvegarde le caractère
                MOV   r2,r12            ; et son compteur
pasMieux:       MOV   r11,r20           ; mettre à jour le précédent caractère
                MVI   r12,#1            ; son compteur à 1
                JNZ   r20,loop          ; et on reboucle si pas fini
fin:
```

2. On utilise le tableau d'octets pour compter le nombre d'occurrences de chaque caractère ; ensuite on parcourt ce tableau en cherchant le plus grand élément.

```
loop:           LDB   r20,(r0)          ; charger le caractère
                JZ    r20,suite         ; on a fini de parcourir la chaîne
                ADD   r0,r0,#1          ; avancer le pointeur
                SUB   r20,r20,#'a'      ; mettre r20 entre 0 et 25
```

```
                ADD   r30,r10,r20       ; pointer sur l'octet correspondant
                LDB   r31,(r30)         ; et incrémenter le compteur
                ADD   r31,r31,#1        ; correspondant au caractère
                STB   (r30),r31         ;
                JMP   loop              ; passer au caractère suivant
suite:          MVI   r2,#0             ; sauvegarde meilleur compteur
                MVI   r12,#'a'          ; compteur du caractère courant
suite1:         LDB   r20,(r10)         ; récupérer le compteur
                ADD   r10,r10,#1        ; avancer le pointeur
                SUB   r31,r2,r20        ; compteur >meilleur?
                JGE   r31,pasMieux      ; non, pas mieux
                MOV   r2,r20            ; on met à jour le meilleur compteur
                MOV   r1,r12            ; et le meilleur caractère
pasMieux:       ADD   r12,r12,#1        ; incrémenter le compteur de caractère
                SUB   r31,r12,#'z'      ; a-t-on parcouru tout le tableau?
                JLE   r31,suite1        ; et on reboucle si pas fini
```

**IV. Mémoire cache**

Un programme se compose d'une boucle de 36 instructions à exécuter 3 fois ; les instructions se trouvant, dans l'ordre, aux adresses mémoire 1 à 6, 37 à 48, 7 à 12, 37 à 48. 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, ..., 37-39, 40-42, 43-45, 46-48... 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, 7-12, ..., 37-42, 43-48... 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, 10-18, ..., 37-45, 46-54... 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, ..., 37-54... 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 ?

Corrigé :

1.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $1 \to 3$ | $M + 2C$ | bloc 1 | $1 \to 3$ | $M + 2C$ | bloc 1 | $1 \to 3$ | $M + 2C$ | bloc 1 |
| $4 \to 6$ | $M + 2C$ | bloc 2 | $4 \to 6$ | $M + 2C$ | bloc 2 | $4 \to 6$ | $M + 2C$ | bloc 2 |
| $37 \to 39$ | $M + 2C$ | bloc 3 | $37 \to 39$ | $3C$ | bloc 3 | $37 \to 39$ | $3C$ | bloc 3 |
| $40 \to 42$ | $M + 2C$ | bloc 4 | $40 \to 42$ | $3C$ | bloc 4 | $40 \to 42$ | $3C$ | bloc 4 |
| $43 \to 45$ | $M + 2C$ | bloc 5 | $43 \to 45$ | $3C$ | bloc 5 | $43 \to 45$ | $3C$ | bloc 5 |
| $46 \to 48$ | $M + 2C$ | bloc 6 | $46 \to 48$ | $3C$ | bloc 6 | $46 \to 48$ | $3C$ | bloc 6 |
| $7 \to 9$ | $M + 2C$ | bloc 1 | $7 \to 9$ | $M + 2C$ | bloc 1 | $7 \to 9$ | $M + 2C$ | bloc 1 |
| $9 \to 12$ | $M + 2C$ | bloc 2 | $9 \to 12$ | $M + 2C$ | bloc 2 | $9 \to 12$ | $M + 2C$ | bloc 2 |
| $37 \to 39$ | $3C$ | bloc 3 | $37 \to 39$ | $3C$ | bloc 3 | $37 \to 39$ | $3C$ | bloc 3 |
| $40 \to 42$ | $3C$ | bloc 4 | $40 \to 42$ | $3C$ | bloc 4 | $40 \to 42$ | $3C$ | bloc 4 |
| $43 \to 45$ | $3C$ | bloc 5 | $43 \to 45$ | $3C$ | bloc 5 | $43 \to 45$ | $3C$ | bloc 5 |
| $46 \to 48$ | $3C$ | bloc 6 | $46 \to 48$ | $3C$ | bloc 6 | $46 \to 48$ | $3C$ | bloc 6 |

Soit un total de $16M + 92C$.

2.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $1 \to 6$ | $M + 5C$ | bloc 1 | $1 \to 6$ | $M + 5C$ | bloc 1 | $1 \to 6$ | $M + 5C$ | bloc 1 |
| $37 \to 42$ | $M + 5C$ | bloc 2 | $37 \to 42$ | $6C$ | bloc 2 | $37 \to 42$ | $6C$ | bloc 2 |
| $43 \to 48$ | $M + 5C$ | bloc 3 | $43 \to 48$ | $6C$ | bloc 3 | $43 \to 48$ | $6C$ | bloc 3 |
| $7 \to 12$ | $M + 5C$ | bloc 1 | $7 \to 12$ | $M + 5C$ | bloc 1 | $7 \to 12$ | $M + 5C$ | bloc 1 |
| $37 \to 42$ | $6C$ | bloc 2 | $37 \to 42$ | $6C$ | bloc 2 | $37 \to 42$ | $6C$ | bloc 2 |
| $43 \to 48$ | $6C$ | bloc 3 | $43 \to 48$ | $6C$ | bloc 3 | $43 \to 48$ | $6C$ | bloc 3 |

Soit un total de $8M + 100C$.

3.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $1 \to 6$ | $M + 5C$ | bloc 1 | $1 \to 6$ | $M + 5C$ | bloc 2 | $1 \to 6$ | $M + 5C$ | bloc 1 |
| $37 \to 45$ | $M + 8C$ | bloc 2 | $37 \to 45$ | $M + 8C$ | bloc 1 | $37 \to 45$ | $M + 8C$ | bloc 2 |
| $46 \to 48$ | $M + 2C$ | bloc 1 | $46 \to 48$ | $M + 2C$ | bloc 2 | $46 \to 48$ | $M + 2C$ | bloc 1 |
| $7 \to 9$ | $M + 2C$ | bloc 2 | $7 \to 9$ | $M + 2C$ | bloc 1 | $7 \to 9$ | $M + 2C$ | bloc 2 |
| $10 \to 12$ | $M + 2C$ | bloc 1 | $10 \to 12$ | $M + 2C$ | bloc 2 | $10 \to 12$ | $M + 2C$ | bloc 1 |
| $37 \to 45$ | $M + 8C$ | bloc 2 | $37 \to 45$ | $M + 8C$ | bloc 1 | $37 \to 45$ | $M + 8C$ | bloc 2 |
| $46 \to 48$ | $M + 2C$ | bloc 1 | $46 \to 48$ | $M + 2C$ | bloc 2 | $46 \to 48$ | $M + 2C$ | bloc 1 |

Soit un total de $21M + 87C$.

4.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $1 \to 6$ | $M + 5C$ | bloc 1 | $1 \to 6$ | $M + 5C$ | bloc 1 | $1 \to 6$ | $M + 5C$ | bloc 1 |
| $37 \to 48$ | $M + 11C$ | bloc 1 | $37 \to 48$ | $M + 11C$ | bloc 1 | $37 \to 48$ | $M + 11C$ | bloc 1 |
| $7 \to 12$ | $M + 5C$ | bloc 1 | $7 \to 12$ | $M + 5C$ | bloc 1 | $7 \to 12$ | $M + 5C$ | bloc 1 |
| $37 \to 48$ | $M + 11C$ | bloc 1 | $37 \to 48$ | $M + 11C$ | bloc 1 | $37 \to 48$ | $M + 11C$ | bloc 1 |

Soit un total de $12M + 96C$.

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.