UNIVERSITÉ DE PARIS-DAUPHINE — DU MI2E 2e année

Architecture des ordinateurs (E. Lazard)
Examen du 30 janvier 2008

CORRECTION

(durée 2 heures)

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

1. Représenter 3 et 33 en virgule flottante.
2. Calculer la multiplication virgule flottante de 3 et 33. 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. 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 $1{,}00001_2 \times 2^3 = 8{,}25$.
   Comparer les valeurs obtenues pour chacun des deux calculs suivants effectués en virgule flottante : $(33 - 1) \times 3$ et $33 \times 3 - 3$.

Corrigé :

1. $3 = 1{,}5 \times 2^1 = \boxed{0\,1000\,10000}$
   $33 = 1{,}03125 \times 2^5 = \boxed{0\,1100\,00001}$
2. La multiplication des deux mantisses $1{,}10000_2$ et $1{,}00001_2$ donne $1{,}10001\not1_2$ comme résultat et le dernier bit de la mantisse saute à cause de l'arrondi. Le résultat obtenu est alors $1{,}10001_2 \times 2^{1+5} = 2^6 + 2^5 + 2^1 = 98$.
3. $33 - 1 = 32$ s'écrit exactement $1{,}00000_2 \times 2^5$ qui multiplié par $3 = 1{,}10000_2 \times 2^1$ donne $1{,}10000_2 \times 2^6 = 96$. En revanche, 33×3 s'écrit $1{,}10001_2 \times 2^6$ d'où on soustrait $3 = 1{,}10000_2 \times 2^1 = 0{,}000011_2 \times 2^6$. Cela donne $1{,}01111\not1_2 \times 2^6$ qui vaut $2^6 + 2^4 + 2^3 + 2^2 + 2^1 = 94$. Notons que si nous avions arrondi supérieurement le dernier résultat, nous serions retombés sur la valeur exacte.

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 NAND 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 XOR et ET, simplifier les sorties et représenter le circuit à l'aide de 5 portes logiques.

CORRIGÉ :

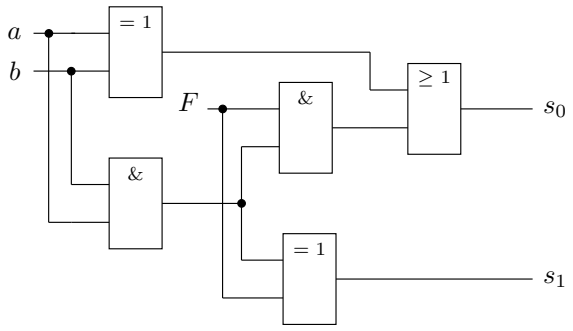
1. Pour $F = 0$

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	1
0	1	1	1
1	0	1	1
1	1	1	0

2. $s_0 = (a \oplus b)\overline{F} + (a + b)F$
   $s_1 = ab\overline{F} + \overline{ab}F = ab \oplus F$
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`. Un tableau de caractères est également stocké en mémoire, chaque caractère sur un octet, chacun à la suite des autres et l'octet qui suit le dernier caractère est nul. L'adresse du premier caractère du tableau se trouve dans le registre `r1`.

   On souhaite compter combien de fois on retrouve dans la première chaîne un des caractères du tableau et mettre ce résultat dans `r2`.

   Par exemple, si la chaîne est `abcbaccdfdeG` et le tableau de caractères contient `acfg`, on veut obtenir comme résultat la valeur 6.

   Écrire une procédure assembleur qui, lorsqu'elle se termine, assure que `r2` contient le nombre de fois qu'on retrouve un caractère du tableau dans la chaîne. On ne vous demande pas de conserver les valeurs des registres à la fin de la procédure.

2. Le tableau contient maintenant des paires de caractères et chaque fois qu'on trouve dans la chaîne un exemplaire du premier caractère de la paire, on veut le remplacer (dans la chaîne originale) par le deuxième caractère de la paire. Par exemple, si la chaîne est `abcbaccdfdeG` et le tableau de caractères contient `aAcCfFgG`, on veut obtenir comme résultat la chaîne `AbCbACCdFdeG` et toujours la valeur 6 comme nombre de caractères trouvés. Modifier votre procédure assembleur pour que `r2` contienne toujours le nombre de fois qu'on retrouve un caractère du tableau dans la chaîne et que les remplacements aient été effectués.. On ne vous demande pas de conserver les valeurs des registres à la fin de la procédure.

Corrigé :

1.
```
            MVI   r2,#0           ; init. compteur
loop:       LDB   r3,(r0)         ; caractère suivant
            JZ    r3, fin         ; fin de la chaîne?
            MOV   r4,r1           ; début du tableau
loop_i:     LDB   r5,(r4)         ; car. suivant du tableau
            JZ    r5,suite        ; fin tableau
            SUB   r31,r5,r3       ; comparer les deux caractères
            JNZ   r31,next        ; différent?
            ADD   r2,r2,#1        ; si non, incrémenter compteur
next:       ADD   r4,r4,#1        ; avancer pointeur tableau
            JMP   loop_i          ; revenir boucle interne
suite:      ADD   r0,r0,#1        ; avancer pointeur chaîne
            JMP   loop            ; revenir boucle chaîne
fin:
```

2.
```
            MVI   r2,#0           ; init. compteur
loop:       LDB   r3,(r0)         ; caractère suivant
            JZ    r3, fin         ; fin de la chaîne?
            MOV   r4,r1           ; début du tableau
loop_i:     LDB   r5,(r4)         ; car. suivant du tableau
            ADD   r4,r4,#1        ; evnt. nv caractère
            JZ    r5,suite        ; fin tableau
            SUB   r31,r5,r3       ; comparer les deux caractères
            JNZ   r31,next        ; différent?
            ADD   r2,r2,#1        ; si non, incrémenter compteur
            LDB   r6,(r4)         ; charger nv caractère
            STB   (r0),r6         ; remplacer dans chaîne
next:       ADD   r4,r4,#1        ; avancer pointeur tableau
            JMP   loop_i          ; revenir boucle interne
suite:      ADD   r0,r0,#1        ; avancer pointeur chaîne
            JMP   loop            ; revenir boucle chaîne
fin:
```

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