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

Architecture des ordinateurs (E. Lazard)
Examen du 26 janvier 2009

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 ; 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. Représenter 0,3125, 1 et 4 en virgule flottante.
2. Quel est le nombre représentable le plus proche de 1/3?
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 $1{,}00001_2 \times 2^3 = 8{,}25$.
   On écrit le code suivant :

```
float f = 1.0/3.0 ;
for (i=0 ; i<100 ; i++)
    f = 4.0*f-1.0 ;
```

   – Si les calculs étaient parfaits, que devrait-on obtenir comme valeur de `f` après la boucle?
   – En prenant comme approximation représentable de 1/3 la valeur calculée à la question 2, quelle est en fait la valeur de `f` après le premier passage dans la boucle?
   – D'après vous, que va-t-il se passer ensuite? Vers quoi va tendre la valeur de `f`?

Corrigé :

1. $0{,}3125 = 1{,}25 \times 2^{-2} = \boxed{0\,0101\,01000}$
   $1 = 1 \times 2^0 = \boxed{0\,0111\,00000}$
   $4 = 1 \times 2^2 = \boxed{0\,1001\,00000}$
2. $1/3 = 0{,}333333... = 1{,}33333... \times 2^{-2}$ est encadré par $1{,}3125 \times 2^{-2}$ et $1{,}34375 \times 2^{-2}$. C'est ce dernier nombre qui est le plus proche. Donc $1/3 \approx 1{,}34375 \times 2^{-2} = \boxed{0\,0101\,01011}$.
3. – Si les calculs étaient parfaits, `f` garderait indéfiniment la valeur 1/3 qui est le point fixe de la fonction $4f - 1$.
   – `f` $= 1{,}01011_2 \times 2^{-2}$ donc 4`f` $= 1{,}01011_2 \times 2^0$ et 4`f` $- 1 = 0{,}01011_2 \times 2^0 = 1{,}01100_2 \times 2^{-2} = 0{,}34375$.

– A cause de l'arrondi supérieur, `f` est légèrement au-dessus de 1/3 et cet écart ne fait que s'accroître à chaque itération car il est multiplié par 4. La valeur de `f` va donc tendre vers plus l'infini...

II. Mémoire cache

Une machine possède un cache de 16 octets. Au cours d'un programme, elle accède successivement les octets situés aux adresses mémoire : 2, 3, 9, 11, 14, 16, 13, 62, 4, 10, 16, 45, 8, 10, 3, 13.

1. On suppose le cache initialement vide et organisé en 16 blocs de 1 octet à accès direct (dans le bloc 1 du cache vont les octets mémoire 1, 17, 33...). Combien sont effectués d'accès cache et d'accès mémoire pour la suite des références mémoire ci-dessus ?
2. On suppose le cache initialement vide et organisé en 4 blocs de 4 octets à accès direct (dans le bloc 1 du cache vont les blocs mémoire 1-4, 17-20, 33-36...). Combien sont effectués d'accès cache et d'accès mémoire pour la suite des références mémoire ci-dessus ?
3. On suppose le cache initialement vide et organisé en 4 blocs de 4 octets à accès associatif (avec l'algorithme de remplacement LRU). Combien sont effectués d'accès cache et d'accès mémoire pour la suite des références mémoire ci-dessus ?

Corrigé :

1. M : 2, 3, 9, 11, 14, 16, 13, 62 (remp. 14), 4, 10, 45 (remp. 13), 8, 13
   C : 16, 10, 3
   Soit 13 accès mémoire et 3 accès cache.
2.

2	*M*	bloc 1 (1-4)	14	*M*	bloc 4 (13-16)	4	*C*	bloc 1	8	*M*	bloc 2 (5-8)
3	*C*	bloc 1	16	*C*	bloc 4	10	*C*	bloc 3	10	*C*	bloc 3
9	*M*	bloc 3 (9-12)	13	*C*	bloc 4	16	*M*	bloc 4 (13-16)	3	*C*	bloc 1
11	*C*	bloc 3	62	*M*	bloc 4 (61-64)	45	*M*	bloc 4 (45-84)	13	*M*	bloc 4 (13-16)

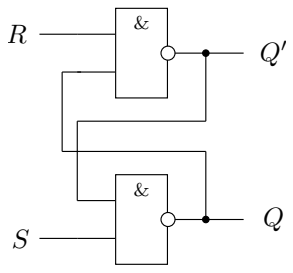
   Soit 8 accès mémoire et 8 accès cache.
3.

2	*M*	bloc 1 (1-4)	14	*M*	bloc 3 (13-16)	4	*C*	bloc 1	8	*M*	bloc 1 (5-8)
3	*C*	bloc 1	16	*C*	bloc 3	10	*C*	bloc 2	10	*C*	bloc 2
9	*M*	bloc 2 (9-12)	13	*C*	bloc 3	16	*C*	bloc 3	3	*M*	bloc 3 (1-4)
11	*C*	bloc 2	62	*M*	bloc 4 (61-64)	45	*M*	bloc 4 (45-84)	13	*M*	bloc 4 (13-16)

   Soit 8 accès mémoire et 8 accès cache.

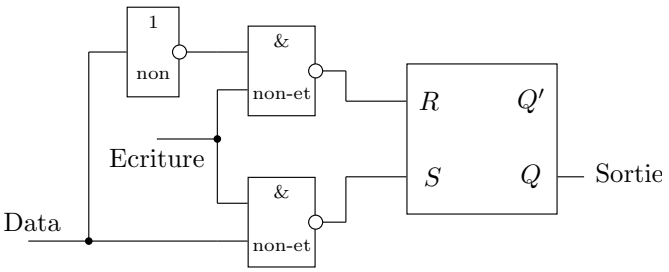
III. Circuits logiques

On considère le circuit suivant composé de deux circuits non-et :



A cause du retour de Q sur R et de Q' sur S, on ne peut pas étudier ce circuit comme d'habitude (ce n'est pas un circuit combinatoire). Il faut maintenant tenir compte du temps, c'est-à-dire calculer un état futur (Q^+) en fonction de l'état présent (Q) et des entrées (R et S).

1. Montrer que si $R = 1$, $S = 1$, alors l'état du circuit est stable, c'est-à-dire que l'état futur des sorties est identique à l'état présent des sorties.
2. Montrer que si $R = 1$, $S = 0$, alors $Q^+ = 1$, $Q'^+ = 0$.
3. Montrer que si $R = 0$, $S = 1$, alors $Q^+ = 0$, $Q'^+ = 1$.
4. Que se passe-t-il si $S = 0$ et $R = 0$?
5. On construit le circuit :



Quel est l'état de la sortie tant que *Ecriture* vaut 0 ? On met un 1 sur *Data* puis, un court instant, on envoie un 1 sur *Ecriture* avant de le faire revenir à 0 ; que se passe-t-il sur la sortie ? Même question si on met un 0 sur *Data*. A quoi peut servir ce circuit ?

Corrigé :

1. Si $Q = 0$, $Q' = 1$ et donc $Q^+ = 0$, ce qui donne $Q'^+ = 1$. L'état est conservé. De même si $Q = 1$.
2. Comme $S = 0$, Q^+ est forcé à 1, ce qui donne $Q'^+ = 0$.
3. Comme $R = 0$, Q'^+ est forcé à 1, ce qui donne $Q^+ = 0$.
4. Comme $S = R = 0$, $Q^+ = Q'^+ = 1$ mais dès que l'une des deux entrées repasse à 1, la sortie correspondante passe à 0. L'état et donc stable mais pas très utile.
5. Si *Ecriture* $= 0$, $S = R = 1$ et rien ne change. Dès que *Ecriture* $= 1$, $R = Data$ et $S = \overline{Data}$, donc $Q^+ = Data$; ensuite *Ecriture* repasse à 0 mais on a toujours $Q^+ = Data$. Le circuit a mémorisé le bit *Data*.

IV. Assembleur

1. Un nombre entier compris entre 0 et 15 est stocké en mémoire sur un octet, son adresse étant dans le registre `r0`. On souhaite générer le caractère hexadécimal (de `0` à `9` ou de `a` à `f`) équivalent et le mettre dans `r2`. Écrire la procédure assembleur correspondante.
2. Un nombre entier compris entre 0 et 32767 est stocké en mémoire sur deux octets, son adresse étant dans le registre `r0`. On souhaite générer une chaîne de quatre caractères hexadécimaux, se terminant par l'octet nul, représentant l'écriture hexadécimal de ce nombre. Cette chaîne sera donc composée de caractères hexadécimaux (chacun sur un octet) et sera stockée à partir de l'adresse contenue dans le registre `r1`. Pour simplifier, on écrira la chaîne « à l'envers » : le caractère représentant les 4 bits de poids faible sera mis à l'adresse pointée par `r1`, le caractère représentant les 4 bits suivant à l'adresse `r1+1`,... , le 4e caractère à l'adresse `r1+3`. Écrire la

procédure assembleur correspondante. *(On pourra récupérer les quatre bits de poids faible d'un registre en effectuant un ET logique entre ce registre et la valeur 15.)*

Corrigé :

1.
```
            LDB   r2,(r0)           ; récupérer le nombre
            JGE   lettre,r2,#10     ; est-ce une lettre?
            ADD   r2,r2,#'0'        ; sinon on construit le car. chiffre
            JMP   fin               ; et c'est tout...
lettre:     ADD   r2,r2,#'a'-10     ; sinon c'est le car. lettre
fin:
```

ou avec une instruction de moins :

```
            LDB   r2,(r0)           ; récupérer le nombre
            ADD   r2,r2,#'0'        ; on construit le car. chiffre
            JLE   fin,r2,#'9'       ; était-ce une lettre?
            ADD   r2,r2,#'a'-'0'-10 ; on construit le car. lettre
fin:
```

2.
```
            MVI   r31,#4            ; 4 symboles à traiter
            LDH   r2,(r0)           ; récupérer le nombre
loop:       AND   r3,r2,#15         ; extraction des 4 bits poids faible
            LLS   r2,r2,#-4         ; décaler pour diviser par 16
            ADD   r3,r3,#'0'        ; on construit le car. chiffre
            JLE   suite,r3,#'9'     ; était-ce une lettre?
            ADD   r3,r3,#'a'-'0'-10 ; on construit le car. lettre
suite:      STB   (r1),r3           ; stocker le caractère
            ADD   r1,r1,#1          ; avancer le pointeur
            SUB   r31,r31,#1        ; décrémenter compteur
            JNZ   r31,loop          ; et revenir si encore des bits
            MVI   r3,#0             ; le zéro final
            STB   (r1),r3           ; à mettre
```