Architecture des ordinateurs (E. Lazard)
Examen du 26 janvier 2015

(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 $2^{-5} = 0{,}03125$).

1. Représenter 0,6875, $\frac{1}{2}$ et 4 en virgule flottante.
2. Quels sont les plus grand et plus petit nombres strictement positifs représentables ?
3. Quels sont les deux nombres représentables encadrant $1/6$? (Donner leur valeur et représentation.)
4. 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 :

Listing 1. Calculs

```
float f = 1.0 / 6.0;
for (i = 0; i < 100; i++)
    f = 4.0 * f - 0.5;
```

   - Si les calculs étaient parfaits, que devrait-on obtenir comme valeur de `f` après la boucle ?
   - En prenant comme approximation représentable de $1/6$ la **borne inférieure** calculée à la question 3, quelle est en fait la valeur de `f` après le premier passage dans la boucle ? Et après un deuxième passage ?
   - D'après vous, que va-t-il se passer ensuite ? Vers quoi va tendre la valeur de `f` ?

Corrigé :

1. $0{,}6875 = 1{,}375 \times 2^{-1} = \boxed{0\,0110\,01100}$
   $\frac{1}{2} = 1 \times 2^{-1} = \boxed{0\,0110\,00000}$
   $4 = 1 \times 2^2 = \boxed{0\,1001\,00000}$
2. Le plus grand nombre représentable est $\boxed{0\,1111\,11111}$ qui est égal à $1{,}11111_2 \times 2^{15-7} = (1 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5}) \times 2^8 = 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 = 504$.
   Le plus petit nombre strictement positif représentable est $\boxed{0\,0001\,00000}$ qui est égal à $1{,}00000_2 \times 2^{1-7} = 2^{-6}$.
3. $1/6 = 0{,}166666... = 1{,}33333... \times 2^{-3}$ est encadré par $1{,}3125 \times 2^{-3}$ et $1{,}34375 \times 2^{-3}$. C'est-à-dire par $\boxed{0\,0100\,01010}$ et $\boxed{0\,0100\,01011}$.

4. – Si les calculs étaient parfaits, `f` garderait indéfiniment la valeur $1/6$ qui est le point fixe de la fonction $4f - 1/2$.
   – La borne inférieure est $0{,}1640625 = 1{,}3125 \times 2^{-3} = \boxed{0\,0100\,01010}$; on a donc $f_0 = 1{,}01010_2 \times 2^{-3}$ donc $4f_0 = 1{,}01010_2 \times 2^{-1}$ et $f_1 = 4f_0 - 1/2 = 0{,}01010_2 \times 2^{-1} = 1{,}01000_2 \times 2^{-3} = 0{,}15625$. L'écart entre f_1 et f_0 est donc de $2^{-4} \times 2^{-3} = 2^{-7}$.
   Au deuxième passage, on a $f_1 = 1{,}01000_2 \times 2^{-3}$ donc $4f_1 = 1{,}01000_2 \times 2^{-1}$ et $f_2 = 4f_1 - 1/2 = 0{,}01000_2 \times 2^{-1} = 1{,}00000_2 \times 2^{-3} = 0{,}125$. L'écart entre f_2 et f_1 est maintenant de $2^{-2} \times 2^{-3} = 2^{-5}$.
   – A cause de l'arrondi inférieur de départ, `f` est légèrement en-dessous de $1/6$ et la fonction ne converge plus vers son point fixe (instable) ; cet écart ne fait que s'accroître à chaque itération car il est multiplié par 4 (f_3 vaut d'ailleurs 0, soit 2^{-3} de moins que f_2). La valeur de `f` va donc tendre vers moins l'infini...

II. Assembleur

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`.

On souhaite savoir si cette chaîne est un palindrome, c'est-à-dire qu'elle est identique quel que soit le sens de lecture (de gauche à droite ou de droite à gauche). Par exemple, `"laval"`, `"abba"` et `"ici ici"` sont des palindromes. On ne distinguera pas les lettres des autres caractères ; autrement dit, il faut absolument que la chaîne soit symétrique caractère par caractère. Ainsi, `"la val"` n'est pas un palindrome car l'espace « ne colle pas » avec le `v`. Une chaîne vide ne sera pas considérée comme un palindrome valide.

Écrire une procédure assembleur qui, lorsqu'elle se termine, met la valeur 1 dans le registre `r1` si la chaîne pointée par `r0` est un palindrome et 0 sinon. On ne vous demande pas de conserver la valeur de `r0` à la fin.

Corrigé :

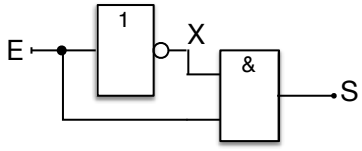
Listing 2. Validation palindrome

```
        MVI     r1,#0
        LDB     r2,(r0)         ; Test si la chaîne est vide
        JZ      r2,fin
        MOV     r10,r0
loop:   ADD     r10,r10,#1      ; parcours de la chaîne
        LDB     r2,(r10)        ; jusqu'au dernier caractère
        JNZ     r2,loop
        SUB     r10,r10,#1      ; sur lequel pointe maintenant r10
cmp:    LDB     r2,(r0)         ; chargement des deux caractères
        LDB     r3,(r10)        ; symétriquement placés
        SUB     r30,r2,r3       ; et comparaison
        JNZ     r30,fin         ; si différent, c'est fini
        ADD     r0,r0,#1        ; décalage des pointeurs
        SUB     r10,r10,#1
        SUB     r30,r0,r10      ; comparaison des deux pointeurs
        JLE     r30, cmp        ; et retour dans la boucle
        MVI     r1,#1           ; ok, le milieu est dépasé
fin:
```

III. Circuits logiques

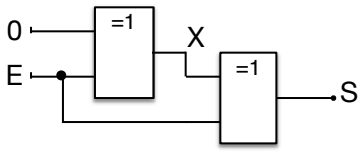
Le temps de passage d'une porte logique est la durée entre l'instant où les signaux sont appliqués à l'entrée et celui où leur effet se répercute en sortie. Jusqu'à présent, ce temps de passage a été ignoré dans un souci de simplification. Toutefois, le temps de passage d'une porte logique n'est jamais nul (de l'ordre de 5 à 25 ns). Il en résulte qu'un changement des données en entrée d'un montage, non seulement mettra un certain temps à se répercuter en sortie, mais pourra en plus provoquer des changements d'état (impulsions) non souhaités à la sortie. De telles impulsions parasites sont appelées hasards logiques.

1. – Exprimer la valeur de sortie S du circuit ci-dessous (composé d'une porte NON et d'une porte ET) en fonction de son entrée E **sans** tenir compte des temps de passage.



   – On prend maintenant en compte les temps de passage. Compléter (sur la feuille annexe) le chronogramme de ce circuit (toutes les portes logiques ont un temps de passage de Δt) en donnant la valeur de X et S au cours du temps. Que remarquez-vous ?

2. – Exprimer la valeur de sortie S du circuit ci-dessous (composé de 2 portes XOR, la première ayant une entrée fixée à la valeur 0) en fonction de son entrée E **sans** tenir compte des temps de passage.

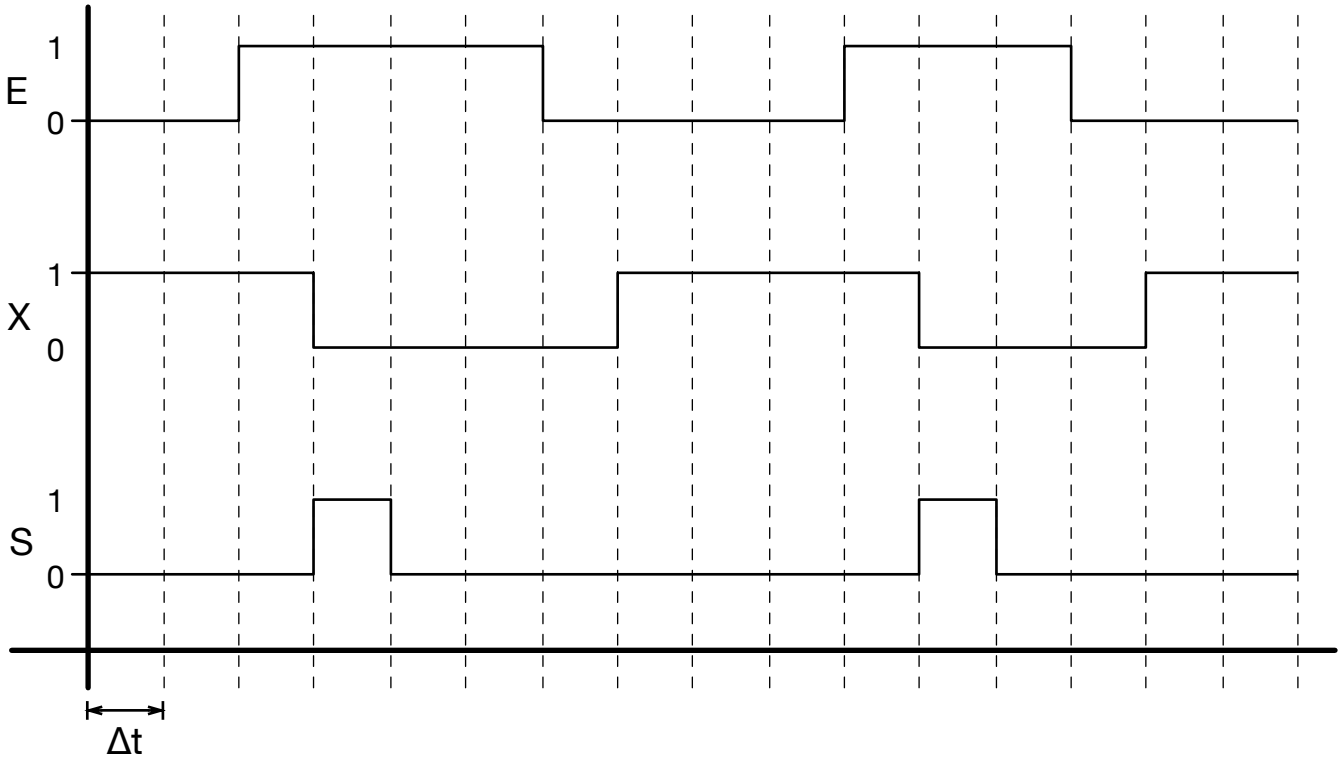


   – On prend maintenant en compte les temps de passage. Compléter (sur la feuille annexe) le chronogramme de ce circuit (toutes les portes logiques ont un temps de passage de Δt) en donnant la valeur de X et S au cours du temps.
   – À quoi ce circuit peut-il bien servir (que détecte-t-il) ?
   – Comment pourrions-nous modifier le circuit pour avoir des impulsions de largeur $3\Delta t$ en S ?

Corrigé :

1.

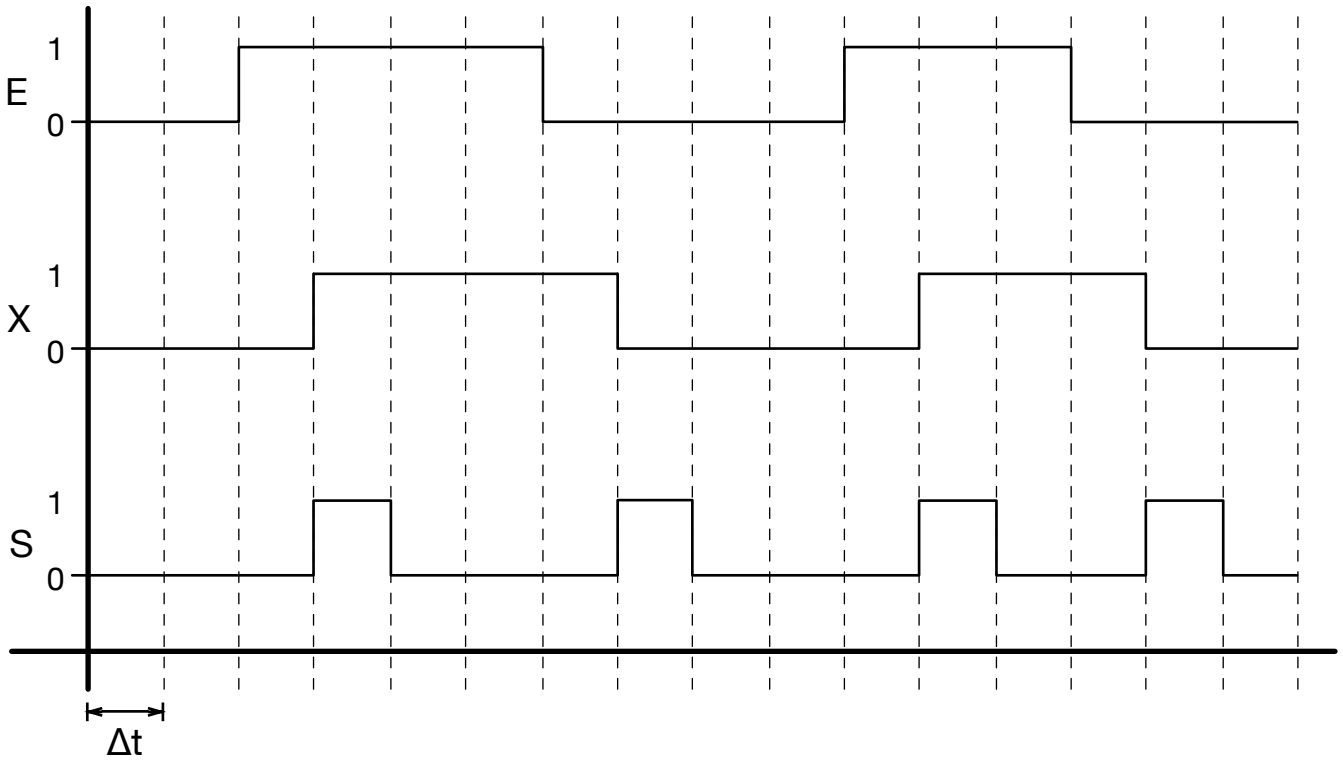
$$S = E.X = E.\overline{E} = 0$$



On voit que S n'est pas constamment à 0 mais au contraire détecte les transitions montantes de E en générant une impulsion.

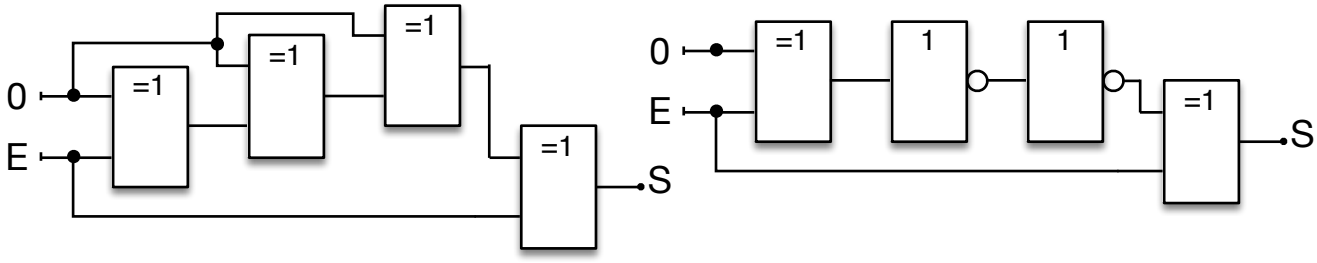
2.

$$S = E \oplus X = E \oplus (E \oplus 0) = E \oplus E = 0$$



Ce circuit génère une impulsion à chaque transition de E, montante ou descendante.

Pour avoir des impulsions de largeur $3\Delta t$ sur S, il faut provoquer un écart de $3\Delta t$ (au lieu de Δt) à l'entrée du deuxième XOR. On peut par exemple introduire deux portes XOR identiques à la première ou deux portes NON entre les deux XOR d'origine.



IV. Mémoire cache

Un programme se compose d'une boucle de 30 instructions à exécuter 5 fois ; cette boucle se trouve aux adresses mémoire 1 à 30. Ce programme doit tourner sur une machine possédant un cache d'une taille de 24 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 6 blocs de 4 instructions. On rappelle que cela veut dire que les mots mémoire 1 à 4, 25 à 28... vont dans le premier bloc, que les mots 5 à 8, 29 à 32... 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 12 blocs de 2 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 deux premiers blocs sont partagés ($1-4$ et $25-28$, $5-8$ et $29-32$), donc :

$$T_1 = 6(M+3C) + M + 3C + M + C = 8M + 22C$$

$$T_2 = 2(M+3C) + 4(4C) + M + 3C + M + C = 4M + 26C$$

$$T = T_1 + 4T_2 = 24M + 126C$$

2. On ne réutilise jamais un bloc, donc :

$$T = 5 \times [7(M+3C) + M + C] = 40M + 110C$$

3. (a) Les trois premiers blocs sont partagés ($1-2$ et $25-26$, $3-4$ et $27-28$, $5-6$ et $29-30$), donc :

$$T_1 = 3(M+C) + 9(M+C) + 3(M+C) = 15M + 15C$$

$$T_2 = 3(M+C) + 9(2C) + 3(M+C) = 6M + 24C$$

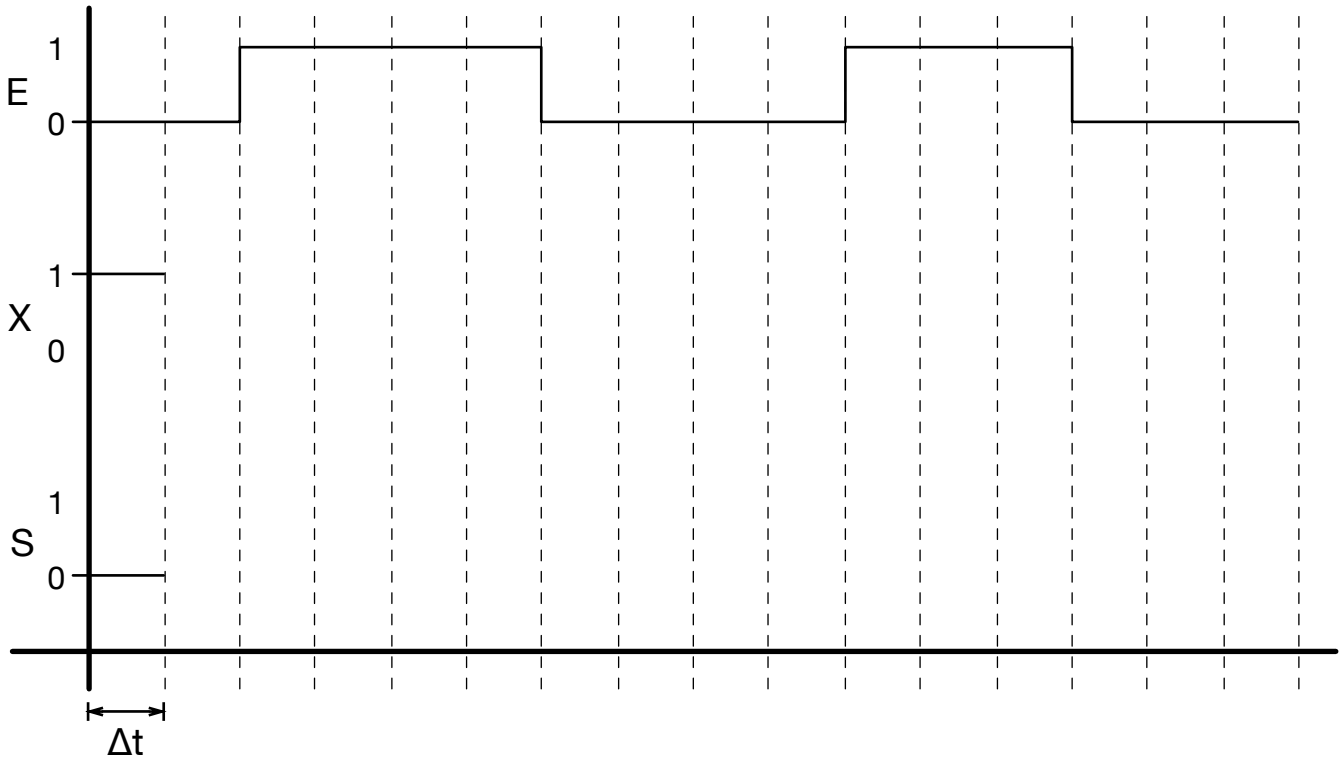
$$T = T_1 + 4T_2 = 39M + 111C$$

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

$$T = 5 \times [15(M+C)] = 75M + 75C$$

N° aléatoire (à choisir et à reporter sur la copie) **:**

III.1 circuit NON et ET



III.2 circuit deux portes XOR