UNIVERSITÉ DE PARIS-DAUPHINE – MIDO L2 MI2E

# Architecture des ordinateurs (E. Lazard)
# Examen du 30 juin 2023

(durée 2 heures)

Documents autorisés : une feuille A4 manuscrite recto-verso
Calculatrice autorisée

## 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 les arrondis s'effectuent ensuite inférieurement lorsque cela est indiqué.**

1. Représenter 0,625 ; 10 et 3,125 en virgule flottante.
2. Calculer les résultats exacts et arrondis des opérations de multiplication flottante $10 \times 3{,}125$ et $10 \times 0{,}625$.
3. On écrit le code suivant :

**Listing 1. Calculs**

```
float f = 3.125;
float g = 0.625;
float x = 10*f - 10*g;
```

   (a) Quelle valeur obtient-on pour `x` si on suppose que tous les calculs sont exacts (sans arrondi) ?

   (b) On suppose qu'après chacune des trois opérations, la valeur obtenue est remise en mémoire *et donc arrondie inférieurement à ce moment-là.* Quelle valeur obtient-on pour `x` ?

   (c) Le compilateur a optimisé la troisième ligne de code qui devient `float x = 10*(f-g);`. Il n'y a plus maintenant que deux arrondis, un après la soustraction et un second après la multiplication. Quelle valeur obtient-on pour `x` ?

   (d) Quelle conclusion en tirez-vous ?

## 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 ET 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 4 portes logiques.

## III. Assembleur

Une chaîne de caractères composée uniquement de lettres majuscules et d'espaces 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`.

Afin de transmettre secrètement cette chaîne, on souhaite décaler chaque lettre (mais pas les espaces) d'une valeur positive contenue dans le registre `r1`, cette valeur étant inférieure à 26. Ainsi, si la chaîne est « `ARCHI TD` » et `r1` contient 4, on veut obtenir la chaîne « `EVGLM XH` ». Le décalage se fait circulairement : après `Z`, on retrouve `A`, puis `B`...

Écrire une procédure assembleur qui, lorsqu'elle se termine, assure que la chaîne initialement pointée par `r0` (dont on n'a pas besoin de conserver la valeur initiale) contient la chaîne d'origine décalée.

## IV. Assembleur

Un tableau d'entiers est stocké en mémoire. Chaque élément a une valeur de 1 à 127 et est donc stocké sur un octet. L'adresse du premier élément du tableau se trouve dans le registre `r0` et le nombre d'éléments dans le registre `r1`.

On souhaite trier ce tableau sur place en ordre croissant par l'algorithme de tri à bulles suivant :

- on parcourt tout le tableau et chaque élément d'indice $i$ est comparé à l'élément d'indice $i + 1$ ;
- si le premier est plus grand que le second (et donc les deux sont inversés pour le tri), on les permute ;
- on effectue la boucle précédente $N$ fois si $N$ est le nombre d'éléments.

Écrire une procédure assembleur qui, lorsqu'elle se termine, assure que le tableau initialement pointé par `r0` est maintenant trié en place. On ne vous demande pas de conserver les valeurs de `r0` et `r1` à la fin.

## V. 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 ?