Nom : _____ **Prénom :** _____

université angers
FACULTÉ DES SCIENCES
Unité de formation et de recherche

Département Informatique
L3 Informatique
Jean-Michel Richer
Architecture des Ordinateurs
jean-michel.richer@univ-angers.fr
2024/2025[1]

Contrôle Continu 2025 2/2

Aucun document (excepté **une** feuille A4 recto/verso), ni outil électronique n'est autorisé. Vous devez soigner votre écriture (la copie ne doit pas être un brouillon) et expliquer clairement votre raisonnement. L'absence de raisonnement ou des étapes de calcul conduit à l'absence de point. En outre, la propreté de la copie, l'orthographe et la grammaire sont prises en compte ainsi que le fait que vous ne rendiez pas votre copie quand cela vous est démandé.

Exercice 1 - 4 pts, 20 min - Soit l'expression suivante : `(x + y) / (x - y)`.

1. donnez une traduction simple en assembleur x86 32 bits de cette expression en utilisant la FPU et en considérant que x et y sont des floats stockés en mémoire

```
section .data
  x: dd 1.5
  y: dd -7
```

2. donnez une traduction simple en assembleur x86 32 bits utilisant la partie basse des registres SSE
3. optimisez le calcul en ne chargeant qu'une seule fois x et y dans la FPU, on rappelle que `fld st2`, par exemple, permet de dupliquer la valeur dans **`st2`** en la rechargeant dans **`st0`** et que l'on peut écrire, par exemple, `fadd st4, st0` qui additionne le contenu de **`st0`** à **`st4`**. Ou encore `fadd st0, st4` qui additionne le contenu de **`st4`** à **`st0`**. Donnez un tableau qui montre ce que contient chaque registre de la FPU après exécution de chaque instruction.

Exercice 2 - 2 pts, 20 min - Soit le code C suivant où `x` est l'abscisse d'une balle (représentée par un caractère) qui rebondit sur un écran texte de 40 caractères (indices 0 à 39). La variable `dx` est la direction dans laquelle se déplace la balle ; `dx` vaut donc 1 ou -1. La fonction `new_dx` donne donc la nouvelle valeur de `dx` en fonction du rebond sur le bord gauche ou droit de l'écran.

```
1 int new_dx(int x, int dx) {
2     return (x == 0) - (x == 39) + ((x >0) and (x<39))*dx;
3 }
```

Donnez une traduction **optimisée** en assembleur x86 32 bits de la fonction en indiquant au préalable quelles associations variables / registres vous utilisez.

Exercice 3 - 8 pts, 50 min - Codage assembleur

```
1  float func(float *x, float *y, float *z, int size) {
2    float prod = 1, sum = 0;
3
4    for (int i = 0; i < size; ++i) {
5      z[i] = x[i] * y[i];
6      prod *= z[i];
7      sum += x[i] + y[i];
8    }
9
10   prod = prod / sum;
11   printf("%f\n", prod);
12
13   return sqrt(prod);
14 }
```

code/cc_mar_exo1.cpp

Note : on considère que les vecteurs `x`, `y` et `z` sont alignés sur des adresses multiples de 16. Le paramètre `size` **n'est pas** multiple de 4

1. donnez le code assembleur x86 32 bits de la fonction `func` en utilisant la FPU. On indiquera au préalable l'association variables / registres.
2. donnez le code C/C++ avec un dépliage par 4 (lignes 2 à 10)
3. donnez une version assembleur vectorisée (SSE) de la fonction en utilisant les instructions `mulps`, `addps`, `subps`, `divps`, `haddps`, `movss` et `shufps` ou `pshufd`. On stockera $prod$ et sum respectivement dans **`xmm0`** et **`xmm1`**, $x[i : i + 3]$, $y[i : i + 3]$ dans **`xmm2`**, **`xmm3`**. On utilisera **`edx`** pour stocker $size$ ou un multiple de $size$.

Exercice 4 - 6 pts, 30 min - Répondre aux questions de cours en expliquant et détaillant votre réponse :

1. rappelez quelles sont les 5 étapes de traitement d'une instruction assembleur
2. qu'est-ce que la prédiction de branchement et à quoi sert-elle ?
3. à quoi sert un registre de segment ?
4. à quoi sert la mémoire cache et comment fonctionne t-elle ?
5. qu'est-ce qu'un nombre auto-descriptif ? Donnez un exemple de nombre auto-descriptif.
6. quelles sont les commandes à utiliser pour produire l'exécutable `a.exe` à partir de `a.asm` ?