# INFO-F-105

Deuxième projet d'assembleur

Année académique 2015–2016

## 1 Introduction

Un automate cellulaire (cellular automaton (en anglais), abbrev. CA) est un modèle discret, à temps discret étudié dans plusieurs domaines scientifiques : informatique théorique, physique, biologie théorique entre autres.

Un automate cellulaire consiste en une grille régulière de *cellules* chacune se trouvant dans un *état* choisi parmi un ensemble fini des valeurs. L'état initial de l'automate (temps $t = 0$) est défini en assignant un état à chaque cellule. Une nouvelle *génération* de l'automate ($t \rightarrow t + 1$) est créée en appliquant simultanément une *règle* à toutes les cellules de la grille. Une *règle* détermine l'état successif d'une cellule en fonction de celui d'un ensemble fini de cellules appelé son *voisinage*.

Dans le cadre de l'amélioration de ses propres outils informatiques, le département de Biologie de l'ULB a contacté le département d'Informatique afin de réaliser un simulateur d'automates cellulaires. En suite à une analyse des besoins du Département de Biologie, nous vous demandons d'implanter le noyau de ce simulateur en langage assembleur, afin d'optimiser la gestion de la mémoire et d'atteindre une efficacité maximale. Ce simulateur que nous vous demandons d'implanter s'appelle *Elementary Cellular Automata* (ECA).

## 2 Description du problème

Plus précisément, nous nous intéressons aux automates cellulaires élémentaires. Un automate cellulaire élémentaire est constitué d'une grille unidimensionnelle (c'est-à-dire un vecteur) de cellules binaires. Le voisinage d'une cellule est constitué par la cellule même et les deux cellules qui lui sont adjacentes (i.e. ses voisins de gauche et de droite). On considère la grille comme étant circulaire, ce qui implique que la première cellule est dans le voisinage de la dernière et inversement.

Avec un voisinage de taille 3, ils existent $2^3$ configurations possibles :

| 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
|---|---|---|---|---|---|---|---|

Si on considère deux états possibles pour chaque cellule, on peut avoir $2^{2^3}$ *règles* différentes pour la création d'une nouvelle génération. Par exemple, la règle $10010010_2 \equiv 146_{10}$ correspond à :

| **Voisinage (t)** | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
|---|---|---|---|---|---|---|---|---|
| **État suivant (t+1)** | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

Si on applique la règle $10010010_2 \equiv 146_{10}$ à une grille de 8 éléments, avec état initial 01001011 on obtient le résultat suivant :

| **État actuel (t)** | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| **État suivant (t+1)** | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

# 3 Description du simulateur ECA

Vous disposerez d'un canevas du simulateur écrit en C++ (ECA.cpp), et d'un canevas du code assembleur (simulation_cell.asm).

Nous vous demandons de :

1. Réaliser la fonction pour calculer l'état suivant d'une cellule en assembleur. La signature de la fonction devra être la suivante :

```
unsigned char simulation_cell(unsigned char left_neighbour, unsigned char cell,
    unsigned char right_neighbour, unsigned char rule[], unsigned char rule_size)
```

   Où :
   — `unsigned char left_neighbour` : Valeur du voisin de gauche
   — `unsigned char cell` : Valeur de la cellule
   — `unsigned char right_neighbour` : Valeur du voisin de droite
   — `unsigned char rule[]` : Vecteur qui contient la règle de génération
   — `unsigned char rule_size` : Taille du vecteur `rule`

2. Compléter le code (C++) de la fonction `simulation_step` afin d'implémenter correctement la mise à jour de toute la grille (i.e. array `world`).

# Indications complémentaires

Vous devez soumettre un fichier compressé `.zip` qui contient un dossier. Ce dossier doit s'appeler `<nom>_<prenom>` (pas de majuscules ni de caractères accentués), le fichier zip doit porter le même nom. Par exemple, Alan Turing doit soumettre un fichier `turing_alan.zip` qui contient le dossier `turing_alan` avec son projet.

Les fichiers à soumettre :
— `ECA.cpp` qui contient le code C++ du simulateur et l'appel à la fonction `simulation_cell` ;
— `simulation_cell.asm` qui contient l'implantation assembleur de la fonction pour calculer l'état futur d'une cellule ;
— `Makefile` qui produit un fichier exécutable `ECA`.

Bon travail !

## Consignes pour la remise du projet

*À respecter scrupuleusement !*

1. Votre projet doit comporter **votre nom** et **votre section**.
2. Votre projet doit être **dactylographié**. Les projets écrits à la main ne seront **pas corrigés**.
3. Votre code doit être **commenté**.
4. Vous devez respecter les modalités suivantes :
   — Rendre une copie papier du projet au secrétariat étudiant.
   — Poster votre fichier avec le projet sur l'UV.
   — Date : **Le lundi 25 avril 2016**
   — Heure : **Avant 10 heures sur l'UV** et **entre 10 heures et 12 heures au secrétariat**

   Après ces heures de remise, les projets sont considérés comme en retard et **ne seront plus acceptés**.