TP 5 – Désassembleur Chip8

Le but de ce TP est de proposer un désassembleur chip8, c'est-à-dire un programme capable de lire des programmes (ROM) pour plateforme à base de chip8 et de proposer le code assembleur représenté sous forme textuelle plutôt que sous forme numérique.
Le chip8 n'est pas un processeur, mais plus l'équivalent d'une machine virtuelle. Néanmoins, son architecture et son mode de fonctionnement sont assez proche de ce que l'on peut trouver dans un processeurs. Évidemment vous écrirez des tests pour toutes les fonctions proposées.

Exercice 1. Représentation d'une machine
Une machine chip8 est constitée (entre autre) :

- d'une mémoire,
- d'un ensemble de 16 registres (notés $V_1, ..., V_F$) de 8bits,
- d'un pseudo-registre d'accés à la mémoire `I` de 12bits,
- d'un "programme counter" (PC) de 16bits,
- le chip8 n'est capable de "comprendre" que des entiers positifs.

Le chip8 est capable d'accéder à 4096 octets de RAM, depuis l'adresse `0x000` (0) à `0xFFF` (4095). Le pseudo-registre `PC` sert à stocker l'adresse de l'instruction en mémoire qui va être exécutée.

1. Proposez une structure capable de représenter une machine Chip8.

Pour les questions suivantes, écrivez systématiquement les tests unitaires associés.

2. Écrivez une fonction qui permet de créer une machine.
3. Écrivez une fonction qui va initialiser une machine, c'est-à-dire mettre tout ses champs aux bonnes valeurs par défaut.

Exercice 2. Chargement d'une ROM
Vous pourrez trouver des ROM pour chip8 un peu partout sur le net. Vérifier juste qu'il sagit bien de ROM pour le chip8 et pas le super chip8.
Le chargement d'une ROM se fait par lecture de la ROM en mode binaire (lecture part octet). Chaque octet est stocké dans la mémoire de la machine à partir de l'adresse `0x200` (contrainte du chip8).

1. Écrivez une fonction qui lit un fichier contenant une ROM et la stocke dans la mémoire d'une machine (rappel : vous utiliserez les fonctions `fopen`, `fread` et `fclose`).
2. Donnez un moyen de stocker le nombre d'octets lues lors du chargement de la ROM.

Exercice 3. Décodage d'une instruction
Une instruction chip8 est codée sur 16bits (2 octets). Par exemple : `0x00E0` représente l'instruction qui demande l'effacement de l'écran. Lorsque l'on veut décoder une instruction, il est donc nécessaire de lire en mémoire à l'adresse pointé par `PC` 2 octets (`char`), de placer les 2 octets dans une seule variable de plus grande capacité.

1. Proposez une fonction qui prend deux char en parmètre et qui retourne le résultat concaténé des 2 `char` dans une variable de type avec une plus grande contenance. Par exemple, avec `a = 0xAB` et `b = 0xCD`, le résultat sera `0xABCD` (vous utiliserez des masques).
2. Écrivez une fonction qui retourne l'instruction présente en mémoire d'une machine à l'adresse indiqué par le pseudo-registre `PC` de la machine.

Une instruction peut être consistuée de certains paramètres : `0x6633` représente le chargement de la valeur `0x33` dans le registre V_6. De manière un peu plus générale, les instructions qui ont la forme `0x6RJJ` représente le chargement d'une valeur `0xJJ` dans le registre V_R.
Nous n'allons pas décoder toutes les instructions chip8 (il en existe 35), mais juste quelques unes (vous pouvez toutes les faires si vous le souhaitez). Le but des fonctions suivante vont être de produire un affichage plus intelligible que `0x6633` en proposant `mov V6, 0x33` (si l'on reprend l'exemple précédent).
Les instructions que nous allons décoder sont les suivantes :

- `0x00E0` → `CLS`
- `0xBjjj` → `JMP jjj`
- `0x6rjj` → `MOV Vr, jj`
- `0x7rjj` → `ADD Vr, jj`
- `0x8ry4` → `ADD Vr, Vy`
- `0xAjjj` → `MOV I, jjj`
- `UNKNOWN` → fonction de fallback si inst pas encore codé

3. Écrivez 4 fonctions : `part1, part2, part3, part4` capable de découper un morceau d'un short passé en entrée. Par exemple : `part1(0x6234)= 0x6000`, `part3(0x6234)= 0x0200`...etc. (vous utiliserez des masques aussi)
4. Proposez une fonction par instruction qui prend l'entier représentant l'instruction en paramètre et propose un affichage de en mode texte (on sait que l'instruction sera la bonne passée en paramètre, pas besoin de tester ici).
5. Proposez une fonction qui exécute la bonne fonction d'affichage en fonction de l'instruction passée en paramètre. si l'instruction n'est pas encore connue, alors la fonction de fallback est appellée.

Exercice 4. Dump de la mémoire
Il faut maintenant mettre tout en commun pour proposer un dump de la mémoire. La boucle de décodage suit la logique suivante :

```
pour i = 0..taille load:
  inst = fetch_inst(machine)
  decode(inst)
  machine.PC = machine.PC + 1
```

1. Codez ce comportement dans une fonction dédiée.
2. Écrivez le `main` terminal qui prend en paramètre de la ligne de commande une ROM à dump.

Exercice 5. S'il vous reste du temps
1. Vous pouvez implémenter l'affichage des autres instructions
2. Vous pouvez aussi code le comportement des instructions et leur impact sur la machine faisant tourner le code (assignation de registres, affichage en mode texte...etc). De cette façon, vous pouvez coder un émulateur Chip8 quasi complet.