I. Choix de la structure de l’émulateur

Pour faciliter la lecture, la compréhension ainsi que la maintenance du code, le choix qui a été fait est de diviser l’émulateur en 5 modules principaux et 2 modules secondaires. Les 2 modules secondaires ont seulement pour objectif d’offrir un certain nombre de fonctions utilitaires telles que le traitement sur les chaînes de caractères (module « str ») pour la lecture des instructions du programme ainsi que l’implémentation d’une liste chaîné (module « linkedList ») pour la représentation de la mémoire dont ous parlerons plus en détail ultérieurement. Enfin les 5 principaux modules ont pour but l’émulation du processeur MIPS à proprement parler. Le module « memory » prendra en charge la représentation de la mémoire offrant comme possibilité l’initialisation de la mémoire, des fonctions d’écriture et de lecture, ainsi qu’un moyen pour écrire sur la sortie standard son état. En ce qui concerne le module « register », il prend en charge la représentation des registres avec les mêmes fonctionnalités que le module « memory ». Le module « instructions », lui, a pour but la traduction des instructions assembleur en leur valeur hexadécimales ainsi que l’exécution de chacune des instructions. Enfin, le module « cpu » s’occupera de la partie exécution du programme avec l’aide de l’ensemble des autres modules. En complément de ces modules, « exceptions » prendra en charge la définition de l’ensemble des exceptions. De même il permettra de fournir un format d’exceptions lisible par l’utilisateur.

II. Choix représentation des registres

Le nombre de registre étant faible, seulement 32 registres d’usage général et 3 registres spécialisés, les 3 registres spécialisés seront stockés simplement dans 3 entiers différents. Quant aux 32 registres d’usage général, ils seront stockés dans un tableau d’entier.

III. Choix représentation de la mémoire

Contrairement à la représentation des registres, il n'est pas réaliste de modéliser la mémoire à l'aide d'un simple tableau. En effet le processeur MIPS avec une mémoire de 4GO, rend cette représentation trop lourde. C'est pour cela qu'une implémentation de liste chaînée sera utilisée. Elle comportera la valeur de l'élément en mémoire, son adresse ainsi que le pointeur vers la prochaine donnée stockée en mémoire. De plus, chacun de ces éléments devra être trié en fonction de son adresse. Cela a pour objectif d'optimiser l'écriture et la lecture. Et ainsi, de garantir de meilleurs performances dans le cas d'une forte utilisation de la mémoire.
Le module « memory » offrira alors une abstraction de cette implémentation de liste. En effet, ce module prendra en charge la lecture et l'écriture en mémoire avec certaines contraintes supplémentaires. Dans un premier temps, seules les valeurs différentes de 0 seront stockées. C'est-à-dire qu'une écriture en mémoire pour une valeur de 0 résultera : soit de l'action vide dans le cas où l'adresse en question n'est pas déjà présente dans la liste, soit à sa suppression dans le cas contraire. De même, une lecture à une adresse n'étant pas présente dans la liste retournera 0. Ce module prendra également en charge la gestion des exceptions liées à l'utilisation de la mémoire (i.e. adresse maximale dépassé ou négative).