Rapport Projet Phase 2 (LDP)

PROPS Thibaut

9 Avril 2025

Les fonctions `parse`

Soit n, la position du mot que l'on veut analyser. On calcule la position des $n+1$ espaces, puis on prend la sous-chaîne de la chaîne d'instruction située entre le $n^{\text{ième}}$ et le $(n+1)^{\text{ième}}$ espaces pour extraire le mot souhaité.

Fonction `is_register()`

On vérifie si le prétendu nom de registre est bien une chaîne de caractères de longueur 1 et que c'est une lettre entre `a` et `d` (c'est-à-dire que sa représentation numérique se situe entre celle de `a` et celle de `d`). Cette fonction permet de savoir, lorsque l'on veut connaître la valeur d'un objet, s'il s'agit d'une `l-value` ou d'une `r-value`.

Fonction `write()`

Étant donné que les cases ne peuvent contenir que des nombres de 8 bits, on va découper le nombre en entrée en deux via la formule :

- `lower8 = value & 0xff` : ne prendre que les 8 premiers bits du nombre.
- `upper8 = value >> 8` : décale le nombre vers la droite. Et vu que nous sommes sur des entiers, cela supprime les 8 premiers bits et transforme les 8 derniers pour obtenir un nombre que l'on peut représenter sur 8 bits.

Ainsi, on écrit dans la première case `lower8` puis dans la deuxième `upper8` (soit une représentation en *Little-Endian*).

Fonction `read()`

On va lire deux nombres 8 bits consécutifs pour reconstruire le nombre 16 bits via la formulation *Little-Endian*.

Fonctions `push()` et `pop()`

On va simplement appeler les fonctions `write()` et `read()` tout en mettant à jour la variable `stack_pointer` afin d'avoir un ADT de pile.