Structures de Données: Simulateur de CPU

Charafeddine EL BOUHALI, Marc-Antoine XIA, Groupe 10

Ce projet a pour but de reproduire les fonctions essentielles d'un cpu d'un point de vue informatique c'est-à-dire permettre d'expérimenter avec la gestion des registres, pile et des segments de mémoire depuis du code assembleur avec une exécution pas à pas. Notre projet offre une base efficace et extensible permettant d'étendre facilement les fonctionnalités comme le set d'instructions.

Résumé:

I. Structure du projet:

- hash.c/hash.h est une table de hachage agrégative offrant un accès rapide à une collection de données tout en laissant l'usager avoir la responsabilité de la gestion de mémoire. Il implémente l'insertion (**hashmap_insert**), la recherche (**hashmap_get**) et la suppression d'éléments (**hashmap_remove**).
- memory.c/memory.h est une structure générale permettant la gestion de système de segment de mémoire pour le cpu. Il implémente la création (**create_segment**) et la suppression de segments (**remove_segment**). Et finalement une fonction pour permettre de trouver un segment convenable pour l'extra segment (**find_free_address_strategy**)
- parser.c/parser.h permet l'analyse et le traitement de code assembleur en texte vers une structure pour pouvoir être lue par le processeur. Il implémente une fonction de lecture (**parse**) et de prétraitement (**resolve_constants**).
- cpu.c/cpu.h implémente les fonctionnalités du processeur lisant les instructions fournies par le parser et les exécutant en mémoire. Il implémente une fonction pour exécuter un programme assembleur (**run_program**) chargé au préalable (**allocate_variables** et **allocate_code_segment**).
- test_… sont les tests relatifs au différentes fonctionnalités du code
- exemple.asm est le fichier test qui sert pour vérifier le bon fonctionnement de toute les instructions supportées
- perf_boucle.asm est un simple programme assembleur avec une grande boucle pour tester la vitesse d'exécution
- perf_compile.asm est un large fichier assembleur avec plein de déclaration de variables et plein d'opérations pour tester la vitesse du parser
- main.c/sdc_main il s'agit du produit final, on doit donner en argument le fichier assembleur à exécuter puis la taille de la mémoire

II. Particularités:

- Notre programme fourni de nombreux code et messages d'erreurs pour pouvoir facilement retrouver les bugs. Ainsi que de nombreux fichiers tests pour tester les différentes composantes du code.

- “hash” est une table de hachage est générique et est de ce fait agrégative, elle laisse la gestion de la mémoire a celui qui en fait usage car on ne connaît pas les détails des variables stockés, si c’est une structure qui comporte par exemple des pointeurs à désallouer. Elle se base sur un tableau fixe de taille 128 et une résolution de collision par probing linéaire. Sa fonction de hachage **simple_hash** est un générateur congruentiel linéaire retrouvé dans les exemples du cours.
- “memory” utilise une liste simplement chaînée ainsi qu’une table de hachage pour gérer ses segments non alloués et alloués respectivement. Lors de la suppression de segment avec **remove_segment** il fusionne les deux segments qui lui sont adjacents.
- “parser” utilise principalement sscanf pour reconnaître les instructions et n’implémente pas de fonctions pour être insensible à la casse et aux espaces
- “cpu” place le segment de la pile “SS” à la fin de mémoire, “DS” le segment de variable just avant et enfin le code “CS” tout au début

III. Performances:

- Parser: Nous avons exécuté le parser dans un test **test_perf_parser.c** avec le fichier **perf_compile.asm** composé des 128 instructions de données et 128 instructions de code sur ma machine nous trouvons les résultats suivants: 100 itérations: 0.021735s, 1000 itérations: 0.178963s, 10000 itérations 1.457945s. Cela nous suggère une progression quasi linéaire en O(n). A titre indicatif cela fait 1783332 lignes traitées par seconde sur ma machine.
- CPU: Nous avons crée un fichier assembleur test comportant boucle d’opérations d’addition, d’empilement, dépilement et comparaison dans **perf_boucle.asm** nous utilisons un main **test_perf_cpu.c** afin de limiter les ralentissements avec l’affichage et nous trouvons les résultats suivants: 100 itérations: 0.011599s, 1000 itérations: 0.106517s, 10000 itérations: 0.818625s. C’est encore quasi linéaire. A titre indicatif cela donne que pour ma machine je peux atteindre 61078 instructions par seconde.

IV. Conclusion:

Nous sommes dans l’ensemble satisfait des performances et de l'efficacité du code. Les défauts sont que l’utilisation très prévalente du void* favorise l’apparition de bugs et la sensibilité du parser à la casse et au espace en fond de la programmation en assembleur assez rigide actuellement.