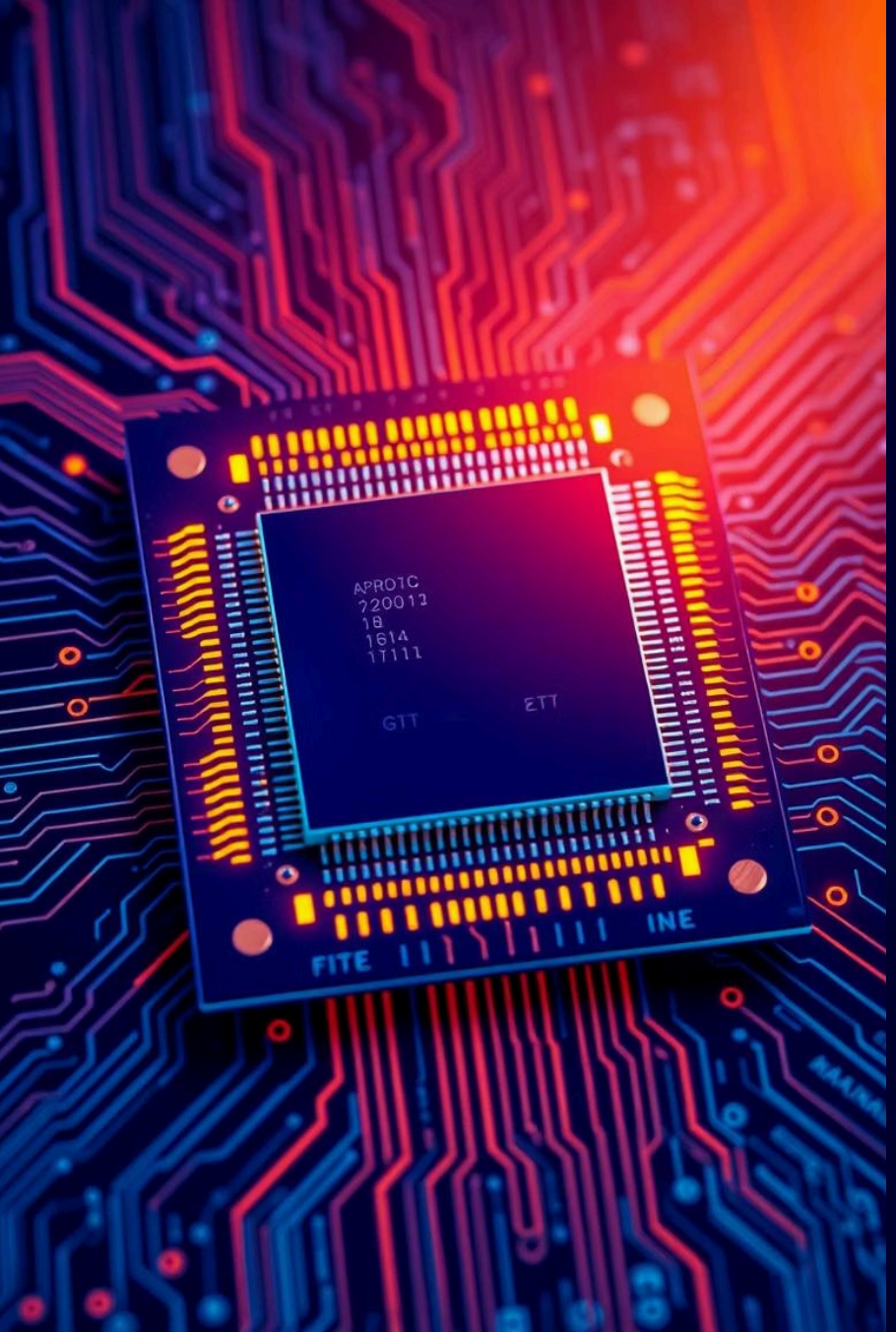


Simulation d'un Processeur (CPU) - Projet LU2IN006

Un simulateur simplifié de CPU en langage C, réalisé dans le cadre du module LU2IN006 : Structures de données.

Objectifs du projet

Comprendre le fonctionnement d'un processeur

Explorer l'architecture et les principes fondamentaux du CPU.

Gérer les composants essentiels

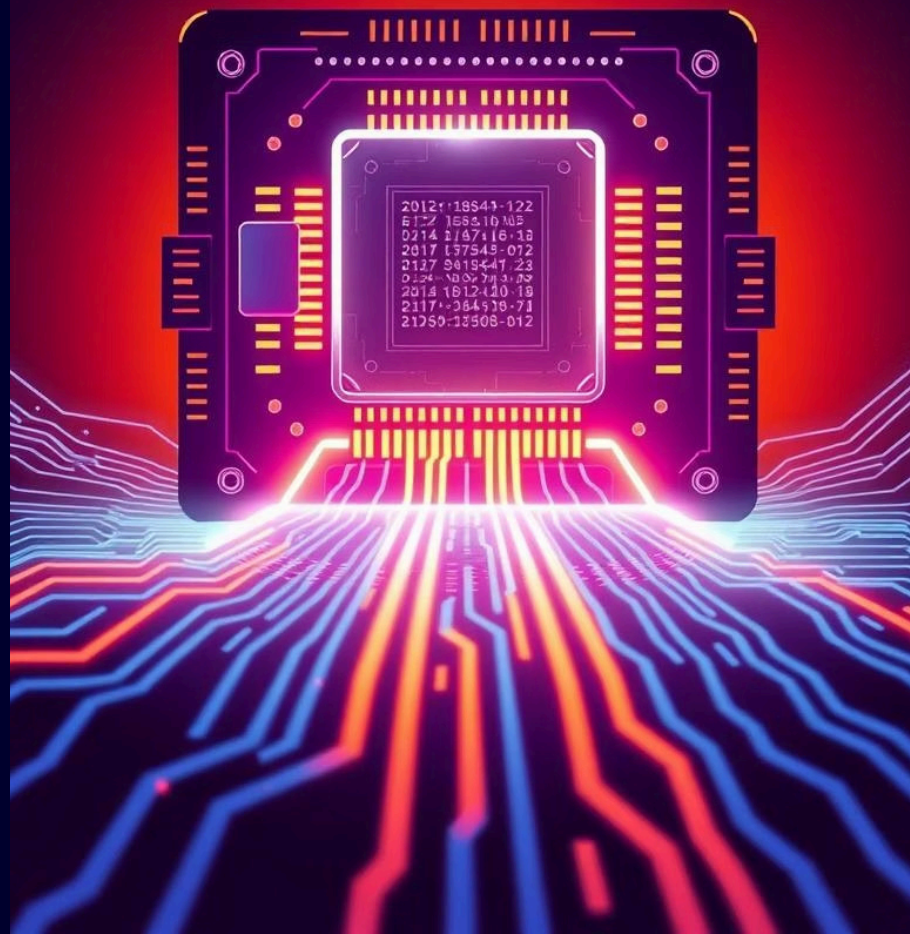
Implémenter la mémoire, les registres et les instructions.

Interpréter le pseudo-assembleur

Traiter les sections .DATA et .CODE du langage.

Implémenter les modes d'adressage

Gérer différents segments mémoire et méthodes d'accès.



Structure du projet

1

Table de hachage générique

Associer des chaînes de caractères à n'importe quel pointeur (void*), avec collision par sondage linéaire.

2

Gestion dynamique de la mémoire

Allouer et libérer dynamiquement des segments de mémoire (ex : segments "DS", "CS", "SS")

3

Parser de pseudo-assembleur

Lire un fichier texte (.DATA et .CODE), analyser les lignes, en extraire des instructions et des labels.

4

CPU simulé

Initialiser un CPU avec des registres et segments mémoire, stocker les données de .DATA dans le segment DS.

5

Résolution d'adressage

Identifier et résoudre les différents types d'opérandes utilisés dans les instructions assembleur, et permettre à handle_MOV de copier les données correctement.

6

Exécution d'un programme assembleur

Exécuter ligne par ligne le code contenu dans le segment .CODE, comme le ferait un vrai CPU.

7

Gestion de la pile (stack segment)

Implémenter un segment "SS" représentant la pile, et les instructions PUSH / POP pour y stocker temporairement des valeurs.

8

Segment supplémentaire dynamique (Extra Segment)

Ajouter un segment dynamique ES qui peut être alloué/libéré à l'exécution via les instructions ALLOC et FREE.

Table de hachage

Structure

HashMap avec HashEntry pour stocker les données.

Adressage ouvert avec probing linéaire.

Applications

- Registres
- Segments alloués
- Pool de constantes
- Labels et variables

Avantages

Accès rapide aux données.

Flexibilité pour différents types d'éléments.

Gestion de la mémoire

Liste chaînée

Organisation des segments libres.

Fonctions

Create, find, remove, destroy.

Table de hachage

Suivi des segments alloués.

Fusion

Optimisation des segments libres adjacents.

Parsing du pseudo-assembleur

Sections

Identification des parties .DATA et .CODE.

Extraction

Conversion en structures Instruction*.

Détection

Identification des labels et variables.

Résultat

Utilisation de ParserResult pour le traitement.

Segments mémoire simulés

CS (Code Segment)

Contient les instructions à exécuter.

SS (Stack Segment)

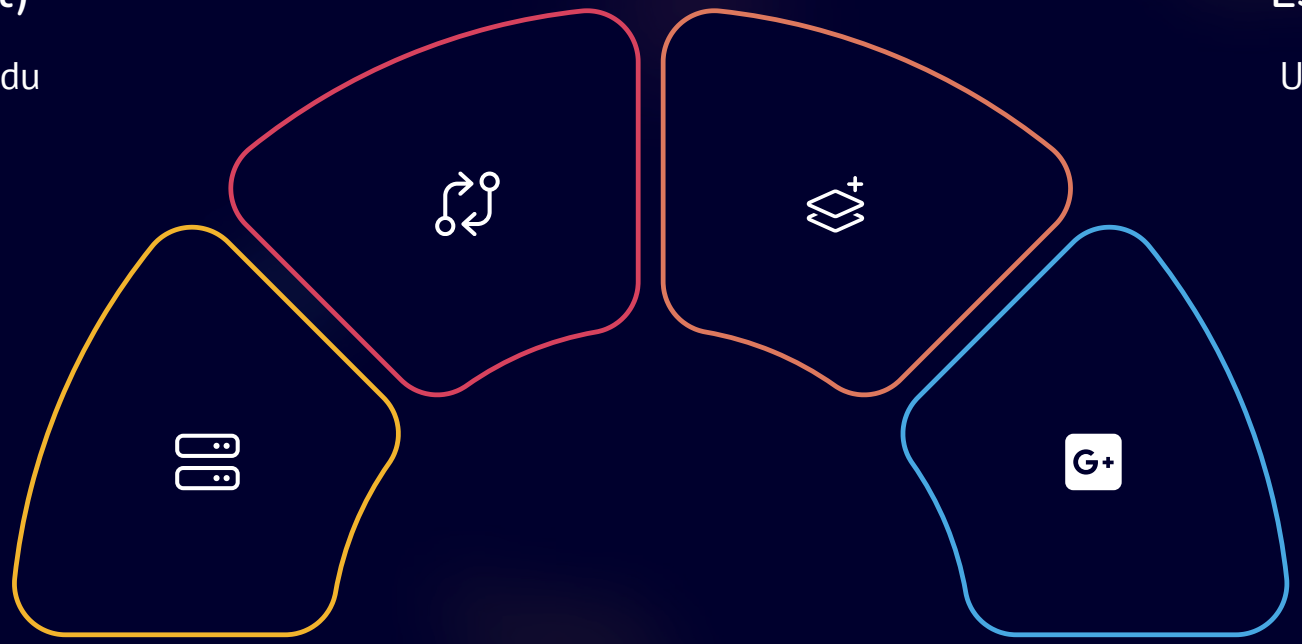
Gère la pile pour PUSH/POP.

DS (Data Segment)

Stockage des variables du programme.

ES (Extra Segment)

Utilisé pour la mémoire dynamique.



Analyse des Performances

Parser Pseudo-Assembleur

Testé avec **perf_compile.asm**. 128 instructions de données et 128 de code.

- 100 itérations: 0.021735s
- 1000 itérations: 0.178963s
- 10000 itérations: 1.457945s

Progression quasi linéaire en O(n). Environ 1 783 332 lignes/seconde.

Simulation CPU

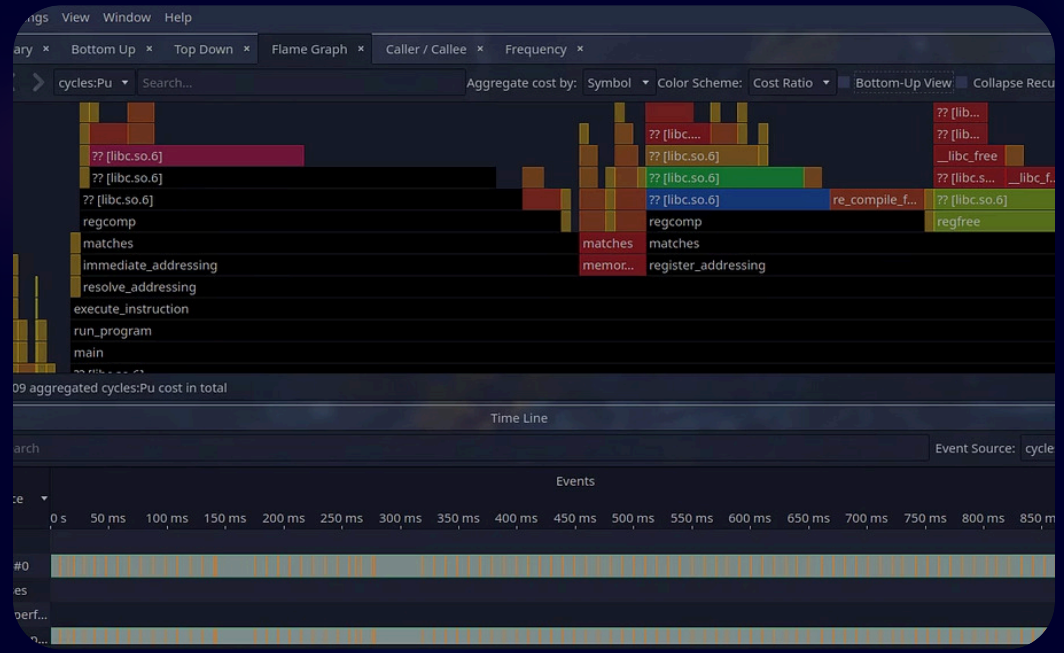
Testé avec **perf_boucle.asm**. Boucle d'additions, empilements, dépilements et comparaisons.

- 100 itérations: 0.011599s
- 1000 itérations: 0.106517s
- 10000 itérations: 0.818625s

Également quasi linéaire. Environ 61 078 instructions/seconde.

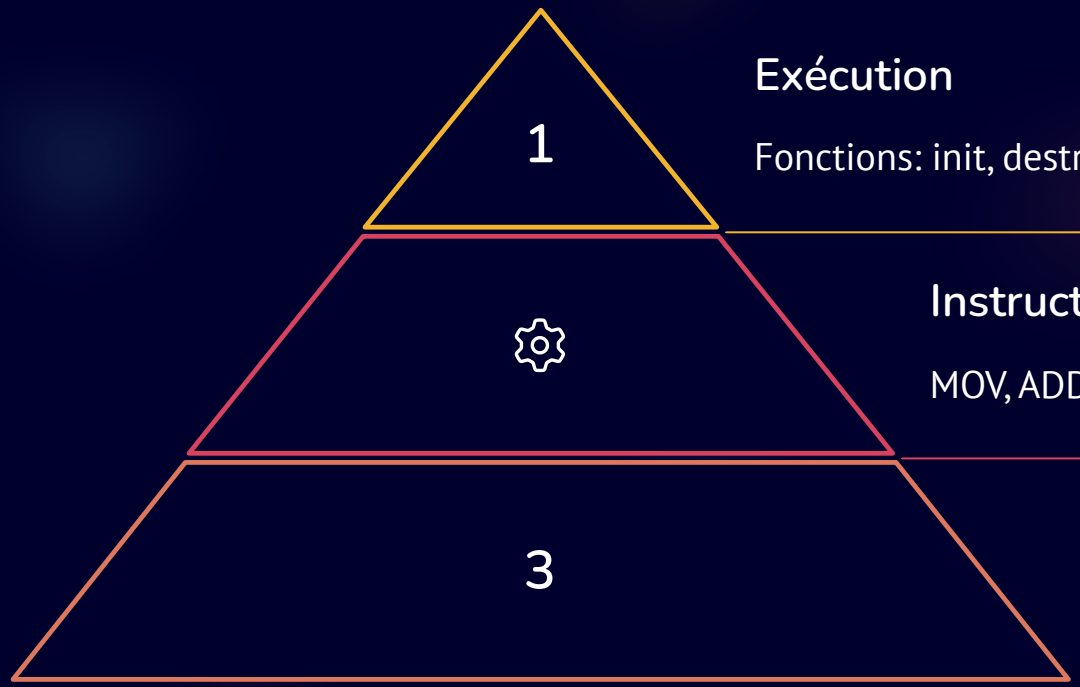
Identification des fonctions les plus coûteuses en temps

Le flamegraph généré à partir de l'exécution du test test_perf_parser.c



Le flamegraph généré à partir de l'exécution du test test_perf_cpu.c

Simulation CPU



Exécution

Fonctions: init, destroy, run, fetch, execute

Instructions

MOV, ADD, CMP, JMP, JZ, JNZ, HALT, PUSH, POP, ALLOC, FREE

Structure CPU

Gestionnaire de mémoire et registres

La structure CPU intègre un gestionnaire de mémoire complet et des registres essentiels: AX, BX, CX, DX, IP, ZF, SF, SP, BP et ES.