Rapport Technique : Simulateur CPU/Mémoire

Saïd Anas
Ben Saïd Nadhir

1 Sujet

Implémentation en C d'un simulateur de processeur comprenant :

- Gestionnaire de mémoire dynamique segmentée
- 4 registres 32-bit (AX, BX, CX, DX)
- Parser pour pseudo-assembleur (.DATA / .CODE)
- 5 modes d'adressage mémoire

---

2 Structures et Architecture

2.1 Fichiers Clés

- `projet.h` : Déclarations des structures et prototypes
- `projet.c` : Implémentations principales
- `main.c` : Programme principal de simulation
- `test_ex1.c`, `test_ex2.c`, etc. : Jeux de tests par fonctionnalité
- `Makefile` : Automatisation de la compilation (`make`)

2.2 Compilation et Exécution

- Compilation : `make`
- Exécution : `./main`

2.3 Fonctions Principales

- **EX1** : Gestion de la table de hachage
- **EX2** : Gestion mémoire segmentée
- **EX3** : Parsing de fichiers assembleur
- **EX4-6** : Simulation complète CPU

---

3 Structures de Données

```
typedef struct {
    char* key;
    void* value;
} HashEntry;

typedef struct segment {
    int start;
    int size;
    struct segment* next;
} Segment;

typedef struct {
    MemoryHandler* memory_handler;
    HashMap* context;
    HashMap* constant_pool;
} CPU;
```

---

4 Algorithmes Clés

4.1 Hachage (Sondage Linéaire)

- Hash : `simple_hash(key) % TABLE_SIZE`
- Résolution : `(index + i) % TABLE_SIZE`
- Gestion des suppressions via `TOMBSTONE`

4.2 Gestion de la Mémoire Segmentée

- Liste chaînée triée
- Fusion automatique des segments libres

4.3 Parsing Assembleur

- Détection de sections .DATA et .CODE
- Extraction de labels, mnémotechniques et opérandes

4.4 Simulation CPU

- Cycle : Fetch → Decode → Execute
- Résolution automatique des modes d'adressage

---

5 Modes d'Adressage Supportés

Mode	Exemple	Résolution
Immédiat	MOV AX, 5	Valeur immédiate dans AX
Par registre	MOV AX, BX	Copie de BX vers AX
Direct mémoire	MOV AX, [0]	Lecture mémoire DS[0]
Indirect par registre	MOV AX, [BX]	Lecture mémoire DS[val(BX)]
Label (saut)	JZ label	Remplacement par adresse code

---

6 Jeux de Tests

6.1 Tests Unitaires

- **test_ex1.c** : Table de hachage (insertion, suppression, recherche)
- **test_ex2.c** : Allocation et libération mémoire segmentée
- **test_ex3.c** : Parsing de fichiers assembleur
- **test_ex4.c** : Gestion segment de données
- **test_ex5.c** : Modes d'adressage
- **test_ex6.c** : Exécution pas à pas

---

7 Performances et Complexités

7.1 Table de Hachage

Fonction	Complexité
simple_hash	O(n)
hashmap_create	O(1)
hashmap_insert	O(n) (pire cas)
hashmap_get	O(1) (moyenne)
hashmap_remove	O(n)
hashmap_destroy	O(n)

7.2 Gestion Mémoire

Fonction	Complexité
create_seg	O(1)
memory_init	O(1)
find_free_segment	O(n)
create_segment	O(n)
remove_segment	O(n)
memory_handler_destroy	O(n)

7.3 Simulation CPU

Fonction	**Complexité**
cpu_init	O(1)
cpu_destroy	O(n)
store/load	O(1)
allocate_variables	O(n×m)
print_data_segment	O(n)
run_program	O(n)

7.4 Modes d'Adressage

Fonction	**Complexité**
immediate_addressing	O(1)
register_addressing	O(1)
memory_direct_addressing	O(1)
register_indirect_addressing	O(1)
handle_MOV	O(1)
resolve_addressing	O(1)

8 Analyse des Performances

8.1 Table de Hachage

Fonction	Complexité	Description
`simple_hash()`	O(n)	Parcourt toute la chaîne
`hashmap_create()`	O(1)	Allocation fixe
`hashmap_insert()`	O(n)	Sondage linéaire sur collision
`hashmap_get()`	O(1)	Accès direct optimal
`hashmap_remove()`	O(n)	Gère les `TOMBSTONE`
`hashmap_destroy()`	O(n)	Parcourt toutes les entrées

8.2 Gestion Mémoire

Fonction	Complexité	Description
`create_seg()`	O(1)	Allocation simple
`memory_init()`	O(1)	Initialisation fixe
`find_free_segment()`	O(n)	Parcourt la liste chaînée
`create_segment()`	O(n)	Recherche + découpe
`remove_segment()`	O(n)	Fusion de segments adjacents
`memory_handler_destroy()`	O(n)	Libère tous les segments

8.3 Parser Assembleur

Fonction	Complexité	Description
`create_instruction()`	O(1)	Allocation simple
`parse_data_instruction()`	O(m)	m = taille de la ligne
`parse_code_instruction()`	O(m)	Idem
`parse()`	O(n×m)	n lignes de m caractères
`free_instruction()`	O(1)	Libération simple
`free_parser_result()`	O(n)	Parcourt les instructions

8.4 Simulation CPU

Fonction	Complexité	Description
`cpu_init()`	O(1)	Initialisation
`cpu_destroy()`	O(n)	Libère mémoire et registres
`store()`, `load()`	O(1)	Accès mémoire direct
`allocate_variables()`	O(n×m)	n variables de m octets
`print_data_segment()`	O(n)	Parcourt les segments

8.5 Modes d'Adressage

Fonction	Complexité	Description
`immediate_addressing()`	O(1)	Vérification regex
`register_addressing()`	O(1)	Accès via hashmap
`memory_direct_addressing()`	O(1)	Accès direct
`register_indirect_addressing()`	O(1)	Registre → mémoire
`handle_MOV()`	O(1)	Copie simple
`resolve_addressing()`	O(1)	Appelle les fonctions précédentes

Auteurs

- Saïd Anas
- Ben Saïd Nadhir