جامعة الحسن الأول
UNIVERSITÉ HASSAN 1ER

كلية العلوم والتقنيات سطات
FACULTÉ DES SCIENCES ET TECNHIQUES SETTAT

Rapport Projet

MOF6809

SIMULATEUR DE MICROPROCESSEUR MOTOROLA 6809

Creé par:

Mohammed ABDELKHALEK
Mohamed AOULICHAK
Oussama AIT MENDIL
Fouad AYYAD

LST-GI

Encadré par:

Pr. Hicham BENALLA

Sommaire

Introduction

1.1 Présentation du Projet

Le projet **MOF6809** est une application logicielle développée en Java visant à simuler le fonctionnement interne du microprocesseur historique Motorola 6809. Ce processeur 8 bits, célèbre pour son jeu d'instructions orthogonal, sert ici de base pédagogique pour comprendre le cycle *Fetch-Decode-Execute*.

L'application fournit une Interface Graphique Utilisateur (GUI) complète permettant de visualiser l'état de la mémoire (RAM/ROM) et des registres internes en temps réel.

Objectif Pédagogique

L'objectif principal est de permettre à l'utilisateur d'écrire du code assembleur, de le compiler, et de l'exécuter pas-à-pas pour observer les changements d'états électriques simulés.

1.2 Périmètre Fonctionnel

Le simulateur se concentre sur un sous-ensemble spécifique d'instructions et de modes d'adressage, optimisé pour la démonstration académique.

1.2.1 Commandes Supportées

Le programme reconnaît et exécute les mnémoniques suivants, identifiés dans la classe Erreurs :

- **Chargement (Load) 8 bits :** LDA, LDB.
- **Chargement (Load) 16 bits :** LDD, LDS, LDU, LDX, LDY.
- **Arithmétique (Addition) :** ADDA, ADDB, ADDD.
- **Arithmétique (Soustraction) :** SUBA, SUBB, SUBD.

1.2.2 Mode d'Adressage

Le projet implémente spécifiquement le **Mode Immédiat**. Ce mode est caractérisé par l'utilisation du symbole # avant l'opérande (ex : `LDA #$10`). La méthode `CheckModeImmediat` assure la validation syntaxique de ce mode.

Architecture Logicielle

2.1 Structure Globale (MVC)

L'application respecte une séparation des préoccupations, divisée en trois paquetages principaux visibles dans l'arborescence du projet [**?**] :

1. **package Main :** Point d'entrée de l'application.
2. **package graphicUserInterface :** Gestion de l'affichage et des événements.
3. **package programMethodes :** Logique métier (CPU, Mémoire, Décodage).

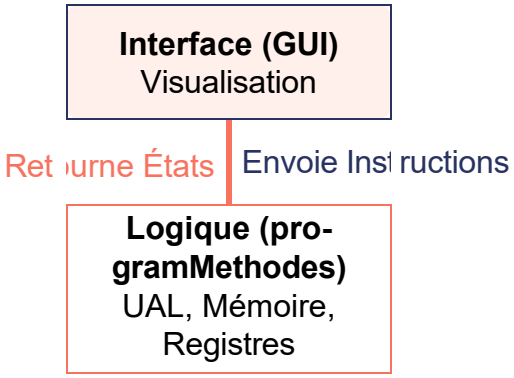

FIGURE – Flux d'interaction simplifié

2.2 Analyse du Package `graphicUserInterface`

La classe principale `GUIClass` étend `Object` mais instancie une `JFrame` nommée "MOF6809" .

2.2.1 Composants Visuels

L'interface est riche et dense, utilisant des `JPanel` pour grouper logiquement les éléments :

— **Visualisation Mémoire :** Deux `JTable` distinctes pour la RAM et la ROM, intégrées dans des `JScrollPane`.
— **Registres :** Chaque registre (A, B, X, Y, S, U, PC, DP) dispose de son propre panneau et champ texte (ex : `ATextField`, `XTextField`).
— **Contrôles :** Des boutons `Compile`, `Execute All`, `Step By Step` et `Reset` permettent le pilotage.

2.2.2 Code Couleur et Design

L'application utilise une palette spécifique définie dans le code :
— Fond des panneaux : Couleur Hex `0xff8d76` (Saumon/Orange) .
— Boutons : Couleur Hex `0xf9745f`.
— Police : Utilisation de "Bahnschrift" pour les titres et "Azeret Mono" pour les données hexadécimales.

Logique de Simulation (Backend)

Ce chapitre détaille le fonctionnement interne du paquet programMethodes, véritable cœur du simulateur.

3.1 Modélisation de la Mémoire

La classe Memoire gère l'espace d'adressage.

— **ROM (Read-Only Memory) :** Initialisée par défaut avec la valeur 0xFF sur une plage de 50 emplacements, commençant à l'adresse 0xFC00. C'est ici que le programme utilisateur compilé est stocké.

— **RAM (Random Access Memory) :** Initialisée à 0x00.

Extrait : Initialisation ROM

```
public int[] AdressROM() {
    int[] AdressROM = new int[50];
    for(int i=0; i<50; i++) {
        AdressROM[i] = 0xFC00 + i;
    }
    return AdressROM;
}
```

3.2 Le Processus de Décodage

La classe Decodage agit comme un traducteur entre l'assembleur et le langage machine. La méthode toOpCodeImmediat transforme les mnémoniques en OpCodes hexadécimaux :

— LDA → "86"

— LDB → "C6"

— LDS → "10CE" (Instruction sur 2 octets)

— ADDA → "8B"

— SUBD → "83"

3.3 L'Unité Arithmétique et Logique (UAL)

La classe `UAL` est la plus complexe. Elle exécute les calculs et met à jour les drapeaux du registre de condition (CCR).

3.3.1 Gestion des Drapeaux (Flags)

Le simulateur calcule rigoureusement les bits du registre CC après chaque opération :

- **C (Carry) :** Retenue, calculée par `CheckCarry8` ou `16`.
- **H (Half-Carry) :** Demi-retenue pour l'arithmétique BCD.
- **Z (Zero) :** Si le résultat est nul.
- **N (Negative) :** Si le bit de poids fort est à 1.
- **V (Overflow) :** Débordement de capacité en complément à deux.
- **P (Parity) :** Si le resultat est paire ou impaire.

3.3.2 Exécution de l'Instruction ADDA (Exemple)

Lorsqu'une instruction `ADDA` est rencontrée : 1. L'opérande est récupéré depuis la ROM. 2. Il est ajouté au contenu actuel du registre A. 3. Le résultat est tronqué à 8 bits (`& 0xFF`). 4. Tous les drapeaux (C, H, Z, N, V, P) sont recalculés et stockés dans l'historique

Flux de Données et Contrôle

4.1 Cycle de Vie d'une Instruction

Le diagramme suivant illustre le cheminement d'une instruction saisie par l'utilisateur, par exemple LDA #$0A.

1. **Saisie :** L'utilisateur tape le code dans TextCodeArea.
2. **Compilation (Bouton Compile) :**
   — Le texte est découpé par la classe Instructions.
   — La classe Erreurs vérifie la syntaxe (présence du #, format hexadécimal valide) .
   — Si aucune erreur n'est détectée, Memoire.ValeurROM encode l'instruction (OpCode 86 + Valeur 0A).
3. **Exécution (Bouton Execute All) :**
   — La méthode UAL.Calcule parcourt la mémoire ROM.
   — Elle détecte 86 (LDA), charge 0A dans le registre A simulé.
   — Les listes d'historique valeursA sont mises à jour.
4. **Affichage :**
   — L'interface récupère la dernière valeur de valeursA.
   — ATextField affiche "0A".
   — Les bits du panneau CC (CC1 à CC8) s'allument ou s'éteignent selon les drapeaux.

4.2 Gestion des Erreurs

Le simulateur est robuste grâce à une vérification stricte en amont. La méthode CheckSyntaxErrorsImmediat s'assure que :

— L'instruction existe.
— L'opérande correspond à la taille attendue (8 bits pour LDA, 16 bits pour LDS).
— Le format est bien hexadécimal (0-9, A-F).

En cas d'erreur, un message précis est affiché dans la zone ERRORArea (ex : "Error At line X") .

Conclusion et Perspectives

5.1 Bilan Technique

Le projet **MOF6809** constitue une implémentation fonctionnelle et visuellement aboutie d'un simulateur de processeur. L'utilisation de Java Swing a permis de créer une interface dense, capable d'afficher simultanément l'état global de la machine, ce qui est un atout pédagogique majeur.

L'architecture orientée objet, avec une séparation claire entre l'interface (GUIClass) et le moteur de calcul (UAL, Memoire), facilite la maintenance et l'évolution du code.

5.2 Limitations et Contraintes Temporelles

Bien que l'architecture logicielle soit conçue pour supporter l'ensemble du jeu d'instructions du Motorola 6809, la version actuelle se concentre exclusivement sur le mode immédiat.

Note Académique

Cette limitation est la conséquence directe des contraintes de temps inhérentes au cadre universitaire du projet. L'implémentation des modes d'adressage direct, étendu ou indexé demanderait une logique de décodage plus complexe (accès RAM en lecture/écriture) qui, bien que prévue dans la structure (méthodes AdressRAM), n'a pas pu être finalisée dans cette itération.

Néanmoins, le socle technique est solide : l'ajout de nouveaux modes ne nécessiterait pas une refonte de l'interface, mais simplement l'extension des classes Decodage et UAL.

Fin du Rapport