Université Hassan 1er
Faculté des Sciences et Techniques Settat

LICENCE SCIENCES ET TECHNIQUES EN GENIE INFORMATIQUE

Simulateur de Microprocesseur Motorola 6809

Encadré par :
Mr. BENALLA Hicham

Réalisé par :
HAMDANI Yasmine
RIADI Yahia
SEHLAOUI Achraf
SENBATI NIZAR
BILAL ZINEDDINE

Aperçu du Motorola 6809

Introduit en 1978, le Motorola 6809 est un microprocesseur 8 bits qui s'est distingué dans l'histoire de l'informatique. Comparé à son prédécesseur, le Motorola 6800, il a apporté des améliorations notables en termes de performance et de capacités. Parmi ses caractéristiques marquantes, on peut citer :

1. **Architecture avancée** : Contrairement à de nombreux autres processeurs 8 bits de l'époque, le 6809 disposait d'un jeu d'instructions plus riche et d'une structure de registres plus complexe, offrant ainsi une plus grande flexibilité pour la programmation.
2. **Modes d'adressage :** Le 6809 a introduit un large éventail de modes d'adressage, ce qui était plutôt inhabituel pour les processeurs de sa catégorie à cette époque.
3. **Performance accrue** : Il était considéré comme l'un des microprocesseurs 8 bits les plus puissants en termes de traitement des instructions complexes.

Impact dans l'Industrie :

Le 6809 a été largement adopté pour divers usages, y compris dans les ordinateurs personnels, les systèmes de jeux vidéo et même dans des applications industrielles. Sa capacité à supporter des systèmes d'exploitation plus avancés a joué un rôle crucial dans son succès.

Pourquoi un Simulateur est Nécessaire ?

1. **Compréhension Historique** : Un simulateur permet d'explorer et de comprendre l'architecture et la programmation des premiers microprocesseurs, essentiels à l'histoire de l'informatique.
2. **Développement de Logiciels Rétro** : Il offre une plateforme pour développer et tester des logiciels conçus pour le 6809, sans nécessiter le matériel physique, qui est devenu rare et précieux.
3. **Outil Éducatif** : Les simulateurs servent à enseigner les concepts de base de l'architecture des microprocesseurs et de la programmation bas-niveau, compétences toujours pertinentes dans l'enseignement de l'informatique.

4. **Préservation du Patrimoine Numérique** : Ils jouent un rôle crucial dans la préservation des logiciels classiques, permettant à ces programmes de continuer à fonctionner sur des systèmes modernes.

En somme, le **Motorola 6809** est non seulement un jalon important de l'histoire de la technologie, mais sa simulation est aussi une fenêtre sur le passé de l'informatique, un outil d'apprentissage précieux, et un moyen de préserver un héritage numérique important.

Introduction Générale

Dans le cadre du développement d'un simulateur informatique complexe, plusieurs composants logiciels jouent un rôle crucial pour assurer son fonctionnement harmonieux et efficace. Les fichiers ArchitecteInterne.java, Editeur.java, Menu.java, RAM.java, et ROM.java constituent les éléments fondamentaux de ce simulateur, chacun apportant une fonctionnalité spécifique et essentielle à l'ensemble du système. Ces fichiers, écrits en Java, un langage de programmation orienté objet et largement utilisé, sont conçus pour interagir ensemble dans le but de simuler le comportement d'une architecture informatique spécifique.

1. **ArchitecteInterne.java** : Ce fichier est au cœur de l'interface utilisateur du simulateur. Il gère l'affichage et la mise à jour des informations critiques liées à l'architecture interne du simulateur. En utilisant les éléments de l'interface graphique Java Swing, ArchitecteInterne sert de tableau de bord interactif, offrant aux utilisateurs une vue d'ensemble des processus en cours dans le simulateur.
2. **Editeur.java** : Ce fichier représente l'éditeur de code intégré au simulateur. Il fournit une interface pour écrire et exécuter des scripts ou des instructions qui seront traitées par le simulateur. L'éditeur est équipé de fonctionnalités telles que la mise en évidence de syntaxe et l'exécution pas à pas, facilitant ainsi pour les utilisateurs la création et le débogage de leurs programmes.
3. **Menu.java** : Ce fichier gère la barre de menu du simulateur, offrant un accès rapide et intuitif aux différentes fonctionnalités du logiciel. Il agit comme une interface centrale pour naviguer entre les différents composants du simulateur, tels que l'éditeur de code, les vues de la RAM et de la ROM, et d'autres outils utiles.
4. **RAM**.java et **ROM**.java : Ces fichiers sont responsables de simuler les fonctionnalités de la mémoire vive (RAM) et de la mémoire morte (ROM) du système informatique. Ils offrent une interface pour visualiser et manipuler les données stockées dans ces composantes mémoire, permettant aux utilisateurs de comprendre et d'analyser comment les données sont traitées et conservées dans le simulateur.
5. **Programme.java** : Cette fenêtre pourrait être dédiée à l'affichage ou à la manipulation de programmes

Objectifs du Simulateur Motorola 6809

Le simulateur de microprocesseur Motorola 6809 est conçu avec des objectifs clairs pour fournir une expérience complète et éducative. Voici les principaux objectifs définis pour ce simulateur :

1. Émulation Fidèle du Jeu d'Instructions

- **Reproduire avec précision** toutes les instructions du processeur 6809.
- **Implémenter les modes d'adressage uniques** du 6809 pour une simulation réaliste.

2. Visualisation et Gestion de la Mémoire

- **Simuler la RAM et la ROM,** permettant aux utilisateurs de voir et de modifier leur contenu.
- **Fournir des outils graphiques** pour visualiser facilement l'état de la mémoire.

3. Capacités de Débogage

- **Permettre des points d'arrêt** pour stopper l'exécution et analyser le comportement du programme.
- **Offrir une exécution pas à pas** pour observer les changements après chaque instruction.
- **Proposer des outils pour inspecter et modifier** l'état du processeur et de la mémoire en temps réel.

4. Interface Utilisateur Intuitive

- **Créer une interface facile à utiliser**, adaptée à différents niveaux d'utilisateurs.
- **Afficher clairement les résultats** des programmes et l'état du processeur.

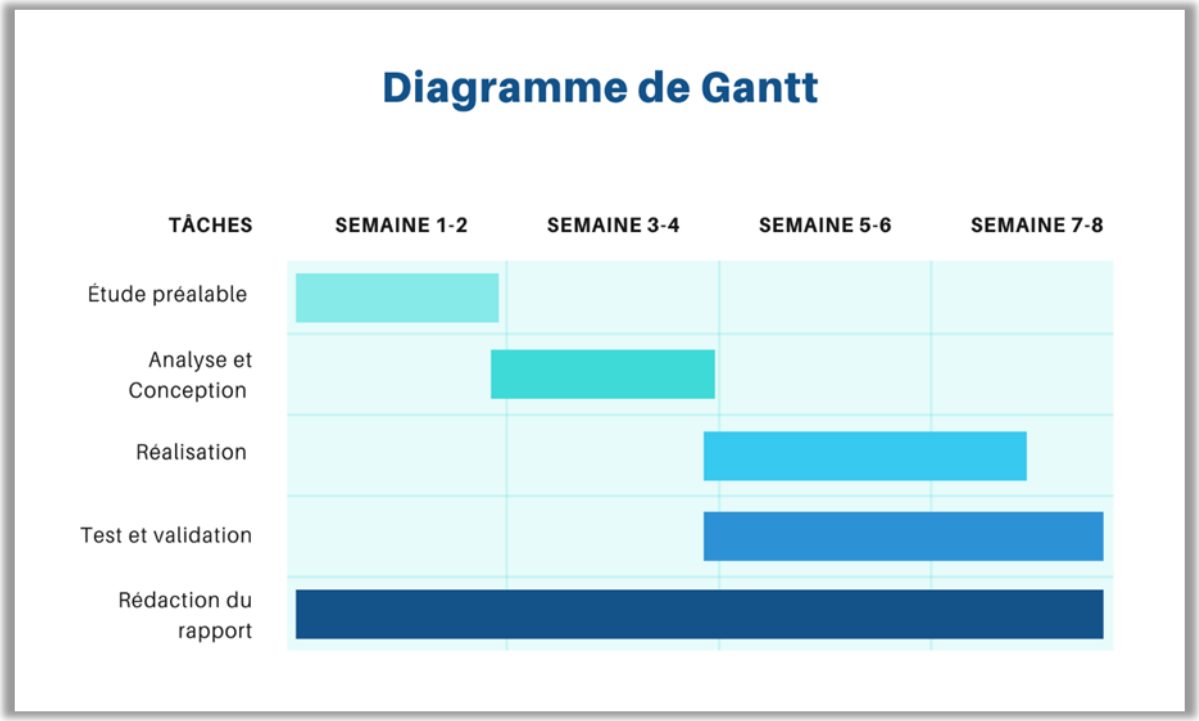
5. Support Éducatif et Documentation

- **Fournir une documentation complète**, expliquant l'utilisation et les fonctionnalités du simulateur.
- **Inclure des ressources pédagogiques** pour enseigner les principes de l'architecture des microprocesseurs.

Diagramme de GANTT

Le diagramme de Gantt est un outil essentiel en gestion de projet, car il permet de visualiser les différentes tâches à effectuer dans le temps. Il s'agit d'un graphique qui montre les différentes étapes du projet, leur durée, leur ordre de réalisation et leurs interdépendances. Grâce à cette représentation, il est possible de suivre l'avancement du projet, de détecter les éventuels retards et de prendre les mesures nécessaires pour les rattraper.

Le diagramme de Gantt ci-dessous illustre la planification et la progression du projet :



Conception et Architecture du Simulateur Motorola 6809

Choix Technologiques :

JAVA :

- **Portabilité** : Java est célèbre pour sa devise "Écrire une fois, exécuter partout", ce qui signifie que le code Java peut s'exécuter sur n'importe quelle plateforme sans modification. Cela est particulièrement utile pour un simulateur destiné à une large audience.
- **Robustesse et Sécurité** : Java offre un environnement de programmation robuste et sécurisé, essentiel pour simuler des architectures de processeur complexes.
- **Outils de Développement** : Java dispose d'un vaste écosystème d'outils et de bibliothèques qui facilitent le développement de simulateurs complexes.

SWING :

- **Interface Utilisateur Graphique (GUI)** : Swing est une bibliothèque GUI puissante pour Java, permettant de créer des interfaces utilisateur riches et réactives. Elle est idéale pour visualiser l'architecture du processeur, l'état de la mémoire et les opérations de débogage.
- **Personnalisation** : Swing offre une grande flexibilité pour personnaliser les composants de l'interface utilisateur, une caractéristique importante pour un simulateur qui nécessite une représentation visuelle spécifique.

Architecture Globale du Simulateur :

Le simulateur est structuré autour de plusieurs composants clés, chacun représenté par un fichier Java différent :

Menu.java :

- **Rôle** : Sert de point d'entrée principal pour l'utilisateur. Il présente les options de l'émulateur et dirige l'utilisateur vers les différentes fonctionnalités.
- **Interaction** : Il interagit avec les autres modules comme l'éditeur, la RAM et la ROM pour lancer des opérations spécifiques.

Editeur.java :

- **Rôle** : Permet aux utilisateurs d'écrire, d'éditer et d'exécuter des programmes conçus pour le Motorola 6809.
- **Interaction** : Communique directement avec la simulation du processeur pour exécuter des instructions et afficher les résultats.

RAM.java et ROM.java :

- **Rôle** : Simulent respectivement la mémoire vive et la mémoire morte du 6809. Ils gèrent le stockage et la récupération des données.
- **Interaction** : Ces modules interagissent étroitement avec l'éditeur et l'architecture interne pour refléter l'état de la mémoire durant l'exécution des programmes.

ArchitecteInterne.java :

- **Rôle** : Gère l'architecture interne du simulateur, y compris la simulation des registres du processeur, l'état actuel du processeur et l'exécution des instructions.
- **Interaction** : Ce composant est au cœur du simulateur, interagissant avec l'éditeur pour l'exécution des instructions et avec les modules RAM/ROM pour la gestion de la mémoire.

Structure du Code :

La structure du projet est organisée de manière à refléter la séparation des préoccupations, où chaque classe a une responsabilité unique :

- **Menu.java** : Interface principale pour la navigation.
- **Editeur.java** : Interface pour la programmation et l'exécution.
- **RAM.java** : Gestion de la mémoire vive.
- **ROM.java** : Gestion de la mémoire morte.
- **ArchitecteInterne.java** : Cœur logique du simulateur, gérant l'état du processeur et l'exécution des instructions.

En combinant ces éléments, le simulateur offre une expérience complète et interactive de l'utilisation et de la compréhension du microprocesseur Motorola 6809.

Problématique :

Comment développer un simulateur de microprocesseur Motorola 6809 qui soit à la fois précis dans son émulation du jeu d'instructions et de l'architecture interne du processeur, facile à utiliser pour l'enseignement et l'apprentissage de la programmation bas-niveau, et suffisamment flexible pour permettre le débogage et la visualisation détaillée de l'état du processeur et de la mémoire ?

Cette problématique englobe plusieurs défis clés :

1. **Fidélité de l'Émulation** : Assurer que le simulateur reproduit fidèlement le comportement du Motorola 6809, notamment son jeu d'instructions unique et ses modes d'adressage.
2. **Interface Utilisateur et Accessibilité** : Créer une interface utilisateur intuitive qui rend le simulateur accessible à des utilisateurs ayant différents niveaux de compétences, notamment des étudiants et des passionnés d'informatique rétro.
3. **Fonctionnalités de Débogage et d'Analyse** : Intégrer des outils de débogage avancés et des capacités d'analyse, comme les points d'arrêt, l'exécution pas à pas, et la visualisation de l'état de la mémoire et des registres.
4. **Performances et Fiabilité** : Assurer que le simulateur est performant et fiable, capable de gérer différentes tâches sans perte de performance ou de précision.
5. **Éducation et Apprentissage** : Fournir un outil qui peut être utilisé comme support pédagogique pour enseigner les concepts fondamentaux de l'architecture des microprocesseurs et de la programmation au niveau système.

En résolvant cette problématique, le projet vise à offrir une ressource précieuse pour l'enseignement, l'apprentissage, et la préservation du patrimoine informatique, en particulier pour ceux qui s'intéressent à l'histoire et au fonctionnement des microprocesseurs classiques.

Description détaillée des composants

Interface Utilisateur

Menu.java : Structure et Options du Menu

Mise en Place du Menu

Dans le fichier Menu.java, le menu du simulateur Motorola 6809 est mis en place en utilisant Java Swing, une bibliothèque pour la création d'interfaces graphiques. Cette interface est le point de départ pour les utilisateurs, leur offrant un accès intuitif aux diverses fonctionnalités du simulateur.

Options du Menu et Actions Associées

1. Ouvrir l'Éditeur :

- Action : Ouvre l'interface de l'éditeur (Editeur.java).
- But : Permet aux utilisateurs de rédiger et de charger des programmes pour le 6809.

2. Gestion de la RAM :

- Action : Lance l'interface de gestion de la RAM (RAM.java).
- But : Permet l'inspection et la modification de la mémoire vive du simulateur.

3. Gestion de la ROM :

- Action : Ouvre l'interface de la ROM (ROM.java).
- But : Permet de visualiser et de manipuler la mémoire morte.

4. Visualiser l'Architecture Interne :

- Action : Affiche l'état actuel de l'architecture interne (ArchitecteInterne.java).
- But : Offre une vue détaillée de l'état interne du processeur.

5. Fonctionnalités de Débogage :

- Action : Active les outils de débogage.
- But : Permet de définir des points d'arrêt et d'exécuter des instructions pas à pas.

6. Aide et Support :

- Action : Fournit de l'aide et de la documentation.
- But : Aide les utilisateurs à comprendre le simulateur et le 6809.

Editeur.java : Fonctionnalités de l'Éditeur

Édition et Exécution de Programmes

- Interface d'Édition : L'éditeur, réalisé dans Editeur.java, fournit une zone de texte où les utilisateurs peuvent écrire ou coller du code destiné au Motorola 6809. Les utilisateurs peuvent sauvegarder, charger ou effacer leurs scripts.

- Exécution de Programmes : Les utilisateurs peuvent exécuter leurs programmes directement depuis l'éditeur. L'exécution simule le comportement du processeur 6809, traitant le code comme s'il était exécuté sur le vrai matériel.

Interaction avec d'Autres Composants

- Interaction avec la RAM et la ROM : Pendant l'exécution, l'éditeur interagit avec les modules de la RAM (RAM.java) et de la ROM (ROM.java) pour simuler l'utilisation de la mémoire par les programmes.
- Visualisation de l'État du Processeur : L'éditeur communique avec ArchitecteInterne.java pour afficher l'état actuel des registres, offrant ainsi un aperçu en temps réel de l'exécution du programme.
- Débogage : L'éditeur permet également l'utilisation de fonctionnalités de débogage, facilitant la mise en place de points d'arrêt et l'exécution pas à pas du code.

En conclusion, Menu.java et Editeur.java jouent des rôles cruciaux dans le simulateur Motorola 6809. Le menu sert de passerelle vers toutes les fonctionnalités principales, tandis que l'éditeur offre un environnement riche pour la programmation, l'exécution et le débogage de programmes destinés au 6809. Ces deux composants, en interaction avec le reste du simulateur, créent une expérience d'émulation complète et pédagogique pour les utilisateurs.

Émulation du Processeur :

ArchitecteInterne.java : Gestion de l'État Interne du Processeur

Fonctionnalités Principales

- Simulation des Registres : Imitation des registres du processeur 6809, comme les accumulateurs et les pointeurs.
- Exécution des Instructions : Interprète et exécute les instructions selon le jeu d'instructions du 6809.
- Gestion des Drapeaux : Suivi des drapeaux de condition (par exemple, zéro, négatif) pour les opérations logiques et arithmétiques.

Affichage et Mise à Jour des Registres

- Visualisation en Temps Réel : Permet de voir l'état actuel des registres, crucial pour comprendre et déboguer le code.
- Mises à Jour Dynamiques : Les registres sont mis à jour à chaque étape d'exécution, reflétant les changements en temps réel.

Interactions

- **Avec l'Éditeur :** Reçoit les instructions de Editeur.java et envoie les résultats de l'exécution.
- **Avec la Mémoire :** Effectue des lectures/écritures dans la mémoire (RAM/ROM).

RAM.java et ROM.java : Émulation de la Mémoire

Émulation de la RAM (RAM.java)

- **Fonctionnement de la Mémoire Vive :** Simule la mémoire RAM pour le stockage et la récupération de données.
- **Structure de Stockage :** Utilise une structure de données pour représenter la RAM, accessible pour les opérations de lecture/écriture.

Émulation de la ROM (ROM.java)

- **Mémoire Morte :** Représente la ROM, stockant des données et instructions en lecture seule.
- **Gestion en Lecture Seule :** Assure que les données dans la ROM ne peuvent pas être modifiées durant l'exécution.

Interaction avec le Simulateur

- **Avec l'Architecture Interne :** Fournissent des données nécessaires pour l'exécution des instructions par ArchitecteInterne.java.
- **Réponse aux Actions de l'Éditeur :** Permettent de visualiser et modifier le contenu de la mémoire via l'éditeur.

En résumé, ArchitecteInterne.java gère le cœur du processeur en simulant ses registres et son exécution d'instructions, tandis que RAM.java et ROM.java s'occupent de l'émulation de la mémoire, jouant un rôle clé dans le stockage des données et l'interaction avec le reste du simulateur.

Émulation du Jeu d'Instructions :

Émulation du Jeu d'Instructions du Motorola 6809

Processus d'Émulation

- Fidélité au Processeur Original : Le simulateur imite le jeu d'instructions spécifique du Motorola 6809. Cela inclut toutes les opérations arithmétiques, logiques, de contrôle de flux et de manipulation de mémoire que le 6809 peut exécuter.

- Interprétation des Instructions : Lorsqu'un programme est exécuté dans le simulateur, chaque instruction est lue, interprétée et exécutée conformément à la façon dont le processeur 6809 le ferait. Ceci est géré principalement par ArchitecteInterne.java.

Exécution et Validation

- Exactitude : L'exécution des instructions suit rigoureusement la logique et le timing du 6809, assurant que les résultats sont aussi proches que possible de ceux obtenus sur un vrai matériel.

- Validation : Des tests et des vérifications sont effectués pour s'assurer que chaque instruction est correctement émulée, en particulier pour les opérations complexes ou moins courantes.

Débogage et Points d'Arrêt

Fonctionnalités de Débogage

- Points d'Arrêt : Les utilisateurs peuvent définir des points d'arrêt dans le code. Lorsque l'exécution atteint un de ces points, elle s'arrête, permettant à l'utilisateur d'inspecter l'état du simulateur à ce moment précis.

- Exécution Pas à Pas : Cette fonctionnalité permet de faire avancer le programme instruction par instruction, offrant un contrôle détaillé et une observation du comportement du simulateur à chaque étape.

Utilité pour les Utilisateurs

- Analyse et Compréhension : Les outils de débogage aident à comprendre comment les instructions affectent l'état du processeur et la mémoire, et sont indispensables pour identifier et résoudre les erreurs dans le code.

Visualisation de la Mémoire

Représentation Visuelle

- Affichage de la RAM et de la ROM : Le simulateur fournit une représentation graphique de la mémoire RAM (RAM.java) et ROM (ROM.java), affichant le contenu de la mémoire sous une forme facilement lisible.
- Mise à Jour en Temps Réel : L'état de la mémoire est mis à jour en temps réel pendant l'exécution des programmes, reflétant les changements dès qu'ils se produisent.

Interaction Utilisateur

- Manipulation Directe : Les utilisateurs peuvent visualiser, modifier ou examiner les données dans la RAM et la ROM. Cela permet une interaction directe avec la mémoire, essentielle pour tester et déboguer des programmes.
- Analyse Détaillée : Cette visualisation aide à comprendre comment les programmes interagissent avec la mémoire, une partie cruciale de la programmation au niveau système.

En résumé, l'émulation du jeu d'instructions du Motorola 6809 dans le simulateur est une réplique précise du comportement du processeur original. Les fonctionnalités de débogage telles que les points d'arrêt et l'exécution pas à pas facilitent l'analyse et la compréhension des programmes. Enfin, la visualisation de la mémoire permet aux utilisateurs d'interagir directement avec la RAM et la ROM, offrant une compréhension approfondie de la gestion de la mémoire dans les systèmes embarqués.

Implémentation Technique

Analyse de ROM.java

Définition de Classe

public class ROM extends JFrame {

- Rôle : La classe ROM étend JFrame, suggérant qu'elle est utilisée pour créer une fenêtre graphique dans l'application. Cette fenêtre est probablement destinée à simuler ou à afficher la mémoire ROM du simulateur.

Constructeur

```
public ROM() {
    setTitle("ROM");
    setAlwaysOnTop(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(750, 120, 212, 276);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    ...
    model.addColumn("Address");
    model.addColumn("Data");
    ...
    for (int i = 5120; i < 6144; i++) {
        String address = intToHex(i); // Convert decimal to hexadecimal
        model.addRow(new Object[]{address, "FF"});
}
```

- Fonctionnement : Ce constructeur initialise la fenêtre ROM. Il définit le titre, la taille, et les propriétés de la fenêtre. Il crée également un modèle de tableau (DefaultTableModel) pour afficher des adresses et des données, probablement pour simuler le contenu de la mémoire ROM.

Méthode Clé

```
public ROM() {
    ...
    for (int i = 5120; i < 6144; i++) {
        String address = intToHex(i); // Convert decimal to hexadecimal
        model.addRow(new Object[]{address, "FF"});
}
```

- Fonctionnement : La méthode semble remplir le modèle de tableau avec des adresses mémoire et des données. Chaque adresse est convertie en hexadécimal et associée à une donnée, ici "FF", ce qui pourrait représenter une valeur par défaut ou un état initial de la mémoire ROM.

Conclusion

La classe ROM semble être conçue pour afficher et manipuler une représentation de la mémoire ROM dans le simulateur. Elle utilise une interface graphique pour montrer les adresses et les données, permettant probablement à l'utilisateur d'interagir ou d'observer l'état de la ROM.

Analyse de la Méthode dans Editeur.java

Méthode : Constructeur Editeur()

```
public Editeur() {
   setTitle("EDITEUR");
   setAlwaysOnTop(true);
   setResizable(false);
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   setBounds(1000, 120, 250, 300);
   contentPane = new JPanel();
   ...
   textArea.setBounds(10, 60, 220, 200);
   contentPane.add(textArea);
   ...
   highlighter = textArea.getHighlighter();
   painter = new DefaultHighlighter.DefaultHighlightPainter(Color.CYAN);
   ...
   JScrollPane scrollPane = new JScrollPane(textArea);
   ...
   new_prog.addActionListener(new ActionListener() { // Bouton pour tout
effacer
      public void actionPerformed(ActionEvent click) {
         ved_all();
         exec.setEnabled(true);
         Pas_a_Pas.setEnabled(true);
}
```

- Fonctionnement : Le constructeur configure la fenêtre principale de l'éditeur. Il définit les propriétés de la fenêtre, ajoute un JTextArea pour la saisie de texte, et y attache un JScrollPane pour le défilement. Un système de mise en

évidence (highlighter) est mis en place, probablement pour mettre en évidence certaines parties du texte. Des actions sont attachées aux boutons pour des fonctionnalités comme effacer le texte (ved_all) et activer ou désactiver certains boutons (exec.setEnabled, Pas_a_Pas.setEnabled).

Conclusion

Ce constructeur établit l'interface utilisateur de base de l'éditeur, en incluant des composants essentiels pour la saisie et la manipulation de texte. Il intègre des fonctionnalités interactives pour améliorer l'expérience utilisateur.

Analyse de la Méthode dans Editeur.java

Méthode : actionPerformed(ActionEvent click)

```
public void actionPerformed(ActionEvent click) {
   ved_all();
   exec.setEnabled(true);
   Pas_a_Pas.setEnabled(true);
}
```

- Fonctionnement : Cette méthode est un gestionnaire d'événements qui réagit aux clics sur certains éléments de l'interface. La méthode ved_all() est appelée, ce qui suggère une réinitialisation ou un nettoyage de l'interface ou du contenu. Les boutons exec et Pas_a_Pas sont réactivés (setEnabled(true)), ce qui implique qu'ils pourraient être désactivés à certains moments et doivent être réactivés après une certaine action (probablement après avoir effacé le texte).

Conclusion

Cette méthode fournit une fonctionnalité interactive pour l'éditeur, permettant à l'utilisateur de réinitialiser l'état de l'éditeur et de réactiver des boutons spécifiques. Cela suggère que l'éditeur a une logique pour gérer l'état de ses composants basée sur les actions de l'utilisateur.
Voulez-vous que je continue avec l'analyse d'autres méthodes dans Editeur.java, si elles sont disponibles ?
D'accord, je vais commencer par examiner Menu.java, puis continuer dans l'ordre que vous avez indiqué. Commençons par extraire et analyser les parties clés de Menu.java. Veuillez patienter un moment.

Analyse de Menu.java

Définition de Classe

public class Menu extends JFrame {

- Rôle : La classe Menu étend JFrame, indiquant qu'elle sert à créer une fenêtre principale pour le menu du simulateur. Cette fenêtre est probablement le point d'entrée principal pour interagir avec les différentes parties du simulateur.

Constructeur

```
public Menu() {
   setResizable(false);
   setFont(new Font("Berlin Sans FB", Font.BOLD, 14));
   setTitle("MOTO-ROLA 6809");
   setBounds(10, 10, 1240, 100);
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   // Création de la barre de menu
   JMenuBar menuBar = new JMenuBar();
   menuBar.setBorder(new EmptyBorder(5, 5, 5, 5));

   // Création du bouton "éditeur"
   JButton editeurItem = new JButton("Editeur");
   editeurItem.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
         frame.setVisible(true);
         if (!edit) {
            frame.setVisible(false);
         ...
}
```

- Fonctionnement : Le constructeur configure les propriétés de la fenêtre du menu, comme sa taille et son style. Il crée une barre de menu (JMenuBar) et y ajoute un bouton pour l'éditeur (JButton). Des gestionnaires d'événements sont attachés aux éléments de la barre de menu pour déclencher des actions, comme rendre visible une autre fenêtre (probablement l'éditeur) en fonction de certaines conditions.

Méthode Clé

```
public Menu() {
   ...
   JButton editeurItem = new JButton("Editeur");
```

```
    editeurItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            frame.setVisible(true);
            if (!edit) {
                frame.setVisible(false);
            ...
}
```

- Fonctionnement : Cette partie du constructeur illustre comment l'application gère les interactions de l'utilisateur avec le menu. L'action de cliquer sur le bouton "Editeur" déclenche une action, comme afficher ou masquer une fenêtre, ce qui suggère une navigation entre différentes parties du simulateur.

Conclusion

La classe Menu semble être la fenêtre principale de l'application, servant d'interface pour accéder aux différentes fonctionnalités du simulateur, comme l'éditeur. Elle utilise une interface graphique simple avec des boutons pour faciliter la navigation de l'utilisateur

Analyse de Programme.java

Définition de Classe

```
public class Programme extends JFrame {
```

- Rôle : La classe Programme étend JFrame, indiquant qu'elle est utilisée pour créer une fenêtre dans le simulateur. Cette fenêtre pourrait être dédiée à l'affichage ou à la manipulation de programmes.

Constructeur

```
public Programme() {
    setTitle("Programme");
    setResizable(false);
    setAlwaysOnTop(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(280, 120, 250, 300);

    contentPane = new JPanel();
    contentPane.setLayout(null);
    setContentPane(contentPane);

    textArea = new JTextArea();
```

```
    textArea.setEditable(false);
    textArea.setBounds(10, 10, 220, 250);
    JScrollPane scrollPane = new JScrollPane(textArea);
    scrollPane.setBounds(10, 10, 220, 250);
    contentPane.add(scrollPane);

    highlighter = textArea.getHighlighter();
    painter = new DefaultHighlighter.DefaultHighlightPainter(Color.CYAN);
}
```

- Fonctionnement : Ce constructeur configure la fenêtre du programme. Il initialise une zone de texte (JTextArea) non éditable, probablement utilisée pour afficher des programmes ou des résultats. Un JScrollPane est ajouté pour permettre le défilement du contenu. Des outils de mise en évidence (highlighter, painter) sont mis en place, ce qui peut indiquer une fonctionnalité pour mettre en évidence certaines parties du texte affiché.

Méthode Clé

```
public Programme() {
    ...
    JScrollPane scrollPane = new JScrollPane(textArea);
    ...
    highlighter = textArea.getHighlighter();
    painter = new DefaultHighlighter.DefaultHighlightPainter(Color.CYAN);
}
```

- Fonctionnement : La méthode illustre comment la fenêtre du programme est configurée pour afficher des informations, avec des fonctionnalités supplémentaires pour améliorer la lisibilité et la présentation du contenu.

Conclusion

La classe Programme semble être conçue pour afficher des informations, telles que des programmes ou des résultats, dans une interface graphique. Elle offre des fonctionnalités de visualisation de texte avec des capacités de mise en évidence pour une meilleure expérience utilisateur.

Analyse de ArchitecteInterne.java

Définition de Classe

```
public class ArchitecteInterne extends JFrame {
```

- Rôle : La classe ArchitecteInterne étend JFrame, ce qui suggère qu'elle est conçue pour créer une fenêtre graphique représentant l'architecture interne d'un simulateur, probablement du processeur 6809.

Constructeur

```
public ArchitecteInterne() {
   setResizable(false);
   setAlwaysOnTop(true);
   setTitle("ARCHITECTURE INTERNE DU 6809");
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   setBounds(10, 120, 226, 420);
   contentPane = new JPanel();
   ...
   PC = new JLabel("0000");
   PC.setFont(new Font("Tahoma", Font.PLAIN, 23));
   ...
   S = new JLabel("0000");
   ...
   U = new JLabel("0000");
   ...
   A = new JLabel("00");
   ...
   [et autres registres]
}
```

- Fonctionnement : Ce constructeur initialise une interface graphique pour afficher divers registres et composants internes du simulateur. Chaque registre (comme PC, S, U, A) est représenté par un JLabel, suggérant une mise à jour dynamique de leur contenu durant l'exécution du simulateur.

Méthode Clé

```
public ArchitecteInterne() {
   ...
   PC = new JLabel("0000");
   ...
   S = new JLabel("0000");
   ...
   U = new JLabel("0000");
   ...
   A = new JLabel("00");
```

```
    ...
    [et autres registres]
}
```

- Fonctionnement : La méthode montre comment l'interface pour l'architecture interne est organisée. Les étiquettes (JLabel) sont utilisées pour représenter les valeurs des registres et autres composants internes du processeur. Cela suggère une interface qui peut afficher et éventuellement mettre à jour l'état interne du simulateur en temps réel.

Conclusion

La classe ArchitecteInterne est responsable de la visualisation de l'architecture interne du simulateur, probablement pour un processeur comme le Motorola 6809. Elle offre une interface utilisateur pour afficher l'état actuel des registres et d'autres composants internes, ce qui est crucial pour comprendre le fonctionnement interne du processeur simulé.

Analyse de RAM.java

Définition de Classe

```
public class RAM extends JFrame {
```

- Rôle: La classe RAM étend JFrame, indiquant qu'elle est conçue pour créer une fenêtre graphique pour représenter la mémoire vive (RAM) dans le simulateur.

Constructeur

```
public RAM() {
    setTitle("RAM");
    setAlwaysOnTop(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(550, 120, 212, 276);
    contentPane = new JPanel();
    ...
    model.addColumn("Address");
    model.addColumn("Data");
    ...
    JTable table = new JTable(model);
    JScrollPane scrollPane = new JScrollPane(table);
    ...
    for (int i = 0; i < 1024; i++) {
```

```
        String address = intToHex(i); // Convertir décimal en hexadécimal
        model.addRow(new Object[]{address, "00"});
}
```

- Fonctionnement: Ce constructeur initialise une interface graphique pour visualiser la mémoire vive. Un modèle de tableau (DefaultTableModel) est utilisé pour représenter les adresses et les données de la RAM. Chaque adresse est convertie en hexadécimal et initialisée avec une valeur par défaut, probablement pour simuler l'état initial de la mémoire vive.

Méthode Clé

```
public RAM() {
    ...
    for (int i = 0; i < 1024; i++) {
        String address = intToHex(i); // Convertir décimal en hexadécimal
        model.addRow(new Object[]{address, "00"});
}
```

- Fonctionnement : Cette boucle dans le constructeur remplit le modèle de tableau avec des adresses mémoire et des données par défaut. Cela montre comment l'interface de la RAM est initialisée et comment les données sont représentées visuellement pour l'utilisateur.

Conclusion

La classe RAM est responsable de la visualisation de la mémoire vive (RAM) dans le simulateur. Elle offre une interface utilisateur pour afficher et éventuellement manipuler les données de la RAM, ce qui est essentiel pour comprendre le fonctionnement de la mémoire dans le simulateur.

Défis Techniques et Solutions :

Défi : Fidélité de l'Émulation : S'assurer que l'émulation du jeu d'instructions et de l'état du processeur est précise.
Solution : Recherche approfondie sur le Motorola 6809 et tests exhaustifs pour valider l'exactitude de l'émulation.

Défi : Performance de l'Interface Utilisateur : Maintenir une interface réactive tout en gérant des opérations complexes en arrière-plan.
Solution : Utilisation de threads pour les opérations longues et optimisation de la mise à jour de l'interface utilisateur.