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.....2

ICON

LOGO

1 Introduction générale

1.1 Contexte du projet : la simulation d'architecture des ordinateurs

L'enseignement de l'architecture des ordinateurs repose sur des concepts abstraits tels que l'exécution séquentielle des instructions, la manipulation des registres, la gestion de la mémoire et l'interprétation des codes machines. Ces notions, bien que fondamentales, demeurent difficiles à appréhender sans une visualisation concrète du fonctionnement interne d'un processeur.

La simulation logicielle constitue ainsi un outil pédagogique privilégié, permettant de représenter dynamiquement le comportement d'une unité centrale de traitement (CPU) sans nécessiter de matériel réel. Dans ce cadre, le projet **MOF6809** propose une approche didactique visant à simuler le microprocesseur Motorola 6809, historiquement reconnu pour la richesse de son architecture et de son jeu d'instructions.

Ce projet s'inscrit pleinement dans un contexte universitaire, avec pour objectif principal de renforcer la compréhension des mécanismes internes d'un processeur à travers une implémentation logicielle fidèle et interactive.

1.2 Présentation du microprocesseur Motorola 6809

Le Motorola 6809 est un microprocesseur 8/16 bits introduit à la fin des années 1970. Il se distingue par une architecture avancée pour son époque, intégrant :

- des registres généraux 8 bits (A et B),
- des registres index 16 bits (X et Y),
- deux pointeurs de pile distincts (S et U),
- un compteur ordinal (PC),
- un registre de condition (CC),
- un registre de page directe (DP).

Cette architecture permet une grande flexibilité dans l'écriture de programmes assembleur et constitue un excellent support pédagogique pour l'étude des notions d'adressage, de pile et de drapeaux conditionnels.

[Image of Motorola 6809 internal register architecture block diagram]

Le projet MOF6809 ne vise pas une reproduction exhaustive du processeur réel, mais une simulation cohérente et fonctionnelle de ses mécanismes essentiels, suffisante pour illustrer le cycle d'exécution d'une instruction et l'impact de celle-ci sur l'état interne du processeur.

1.3 Objectifs du projet MOF6809

Le projet MOF6809 poursuit plusieurs objectifs pédagogiques et techniques :

- Simuler l'exécution d'un sous-ensemble représentatif du jeu d'instructions du Motorola 6809.
- Offrir une interface graphique permettant la visualisation en temps réel :
  - des registres,
  - de la mémoire RAM et ROM,
  - du registre de condition et de ses drapeaux.
- Mettre en œuvre une exécution pas-à-pas afin de suivre précisément le flux d'exécution.
- Illustrer les interactions entre analyse syntaxique, décodage, calcul arithmétique et mise à jour des états internes.

D'un point de vue académique, le projet vise également à structurer un code Java modulaire, organisé en paquetages cohérents, suivant une logique proche du modèle MVC (Modèle–Vue–Contrôleur).

1.4 Structure du présent rapport

Ce rapport est organisé de manière progressive, allant de la présentation générale du projet jusqu'à l'analyse détaillée des mécanismes internes de simulation. Les premiers chapitres définissent le périmètre fonctionnel et l'architecture logicielle. Les chapitres centraux détaillent la conception de l'interface graphique, la gestion de la mémoire, le décodage des instructions et le rôle central de l'unité arithmétique et logique. Enfin, les derniers chapitres dressent un bilan critique du projet, en soulignant les limitations actuelles et les perspectives d'évolution.

2 Périmètre fonctionnel et cahier des charges

2.1 Langage et environnement de développement

Le projet MOF6809 est intégralement développé en langage Java. Ce choix s'explique par la portabilité du langage, sa robustesse et la richesse de ses bibliothèques graphiques. L'interface utilisateur repose sur la bibliothèque Swing, permettant la création de fenêtres, tableaux, boutons et zones de texte interactives.

Le développement a été réalisé dans un contexte académique, sans recours à des frameworks externes, afin de conserver une maîtrise complète des mécanismes internes du programme.

2.2 Jeu d'instructions supporté

Le simulateur implémente un sous-ensemble cohérent du jeu d'instructions du Motorola 6809, réparti en plusieurs catégories fonctionnelles.

2.2.1 Instructions de chargement et stockage

Ces instructions permettent de charger des valeurs immédiates dans les registres internes :

- LDA, LDB (chargement 8 bits),
- LDD (chargement 16 bits),
- LDS, LDU, LDX, LDY.

2.2.2 Instructions arithmétiques

Les opérations arithmétiques sont prises en charge par l'unité arithmétique et logique :

- ADDA, ADDB, ADDD,
- SUBA, SUBB, SUBD,
- MUL.

2.2.3 Instructions logiques

Les opérations logiques binaires incluent :

— ANDA, ANDB,

— ORA, ORB,

— CMPA, CMPB, CMPS,...

2.2.4 Instructions de transfert et d'échange

Les instructions TFR et EXG permettent le transfert ou l'échange de valeurs entre registres, illustrant la flexibilité architecturale du 6809.

2.2.5 Instructions de pile

Les instructions PSHS et PSHU assurent la sauvegarde des registres sur la pile, élément central de la gestion des appels et du contexte processeur.

2.3 Modes d'adressage supportés

Le projet MOF6809 prend en charge deux modes d'adressage principaux :

— le mode immédiat, caractérisé par l'utilisation du symbole #,

— le mode inhérent, caractérisé par l'utilisation du symbole > ou bien rien.

— le mode Etendu, caractérisé par l'utilisation du symbole <.

Ces modes ont été choisis en raison de leur importance pédagogique et de leur simplicité d'implémentation dans un contexte de temps limité.

2.4 Contraintes académiques et limitations temporelles

Le développement du projet s'est déroulé dans un cadre temporel contraint. Cette contrainte a influencé les choix techniques, notamment la limitation volontaire du nombre de modes d'adressage implémentés. L'objectif principal est resté la clarté pédagogique et la stabilité du simulateur, plutôt que l'exhaustivité fonctionnelle.

3 Architecture logicielle globale

3.1 Organisation en paquetages

Le projet MOF6809 est structuré en trois paquetages principaux :

— Main : contient la classe MainClass, point d'entrée du programme.

— graphicUserInterface : regroupe les classes responsables de l'interface graphique et des interactions utilisateur.

— programMethodes : contient le cœur fonctionnel du simulateur (mémoire, décodage, UAL, gestion des erreurs).

Cette organisation favorise la lisibilité du code, la séparation des responsabilités et la maintenabilité du projet.

3.2 Diagramme de classes simplifié

Les classes principales interagissent selon un schéma hiérarchique clair : la classe d'interface appelle les méthodes d'analyse et de décodage, lesquelles invoquent ensuite l'unité arithmétique et logique et la mémoire.

Cette chaîne de responsabilité reflète fidèlement le cycle de traitement d'une instruction dans un processeur réel.

3.3 Rôle du point d'entrée (MainClass)

La classe MainClass constitue le point d'initialisation du projet. Elle instancie l'interface graphique principale et déclenche la mise en place des structures de données nécessaires à la simulation. Aucune logique métier n'est implémentée dans cette classe, conformément aux bonnes pratiques de conception logicielle.

4 Conception de l'Interface Graphique (GUI)

4.1 Philosophie générale du design

L'interface graphique du projet MOF6809 a été conçue comme un outil pédagogique avant tout. Elle ne se limite pas à une simple saisie de code assembleur, mais vise à offrir une visualisation complète et synchronisée de l'état interne du processeur simulé.

Le choix d'une interface dite « vibrante » repose sur l'utilisation de couleurs contrastées et de composants graphiques clairement séparés. Cette approche permet à l'utilisateur, qu'il soit étudiant ou enseignant, d'identifier rapidement les zones fonctionnelles critiques du simulateur.

L'objectif principal est de rendre visibles des mécanismes habituellement invisibles dans un processeur réel, tels que :

— l'évolution des registres après chaque instruction,
— l'impact direct des opérations arithmétiques sur les drapeaux,
— la modification progressive de la mémoire RAM et ROM.

4.1.1 Choix de la palette de couleurs

La palette de couleurs repose principalement sur deux teintes dominantes :

— Violet (`#5a6df0`),
— Violet claire (`#43e4`).

Ces couleurs sont utilisées pour attirer l'attention sur les zones actives du simulateur, notamment les panneaux de registres et les boutons de contrôle. Elles permettent également de distinguer visuellement les éléments statiques (labels, titres) des éléments dynamiques (valeurs mises à jour en temps réel).

4.1.2 Typographie et lisibilité

Deux polices ont été privilégiées :

— **Bahnschrift**, pour les titres et libellés généraux,

— **Azeret Mono Light**, pour l'affichage du code assembleur et des valeurs numériques.

Le recours à une police monospace pour le code et les valeurs hexadécimales améliore la lisibilité et renforce la cohérence visuelle avec les environnements de développement classiques.

4.2 Analyse structurelle de la fenêtre MOF6809

La fenêtre principale du simulateur est organisée en plusieurs zones fonctionnelles distinctes. Cette organisation reflète la structure interne du processeur et facilite la compréhension du flux d'exécution.

4.2.1 Zone de code assembleur et rapport d'erreurs

La zone de saisie du code assembleur constitue le point d'entrée principal de l'utilisateur. Elle permet d'introduire une séquence d'instructions assembleur 6809, lesquelles seront ensuite analysées, décodées et exécutées par le moteur de simulation.

Une zone dédiée au rapport d'erreurs est associée à cette zone de code. Elle affiche les messages générés par la classe `Erreurs`, notamment :

— erreurs de syntaxe,

— opérandes invalides,

— incohérences de taille (8 bits vs 16 bits).

Cette séparation claire entre saisie et diagnostic favorise un apprentissage progressif de l'assembleur.

4.2.2 Visualisation de la mémoire RAM et ROM

La mémoire est représentée graphiquement sous forme de tableaux Swing. Deux tableaux distincts sont utilisés :

— un tableau pour la ROM (mémoire programme),

— un tableau pour la RAM (mémoire de données).

Chaque ligne correspond à une adresse mémoire, tandis que les colonnes affichent les valeurs stockées sous forme hexadécimale. Cette représentation tabulaire permet de suivre précisément l'évolution de la mémoire au fil de l'exécution.

4.3 Panneaux de contrôle des registres

Les registres du processeur sont regroupés dans des panneaux dédiés, séparant clairement les registres 8 bits des registres 16 bits.

4.3.1 Registres 8 bits

Les registres A, B, DP et CC sont affichés individuellement. Leur valeur est mise à jour automatiquement après chaque instruction exécutée. Cette mise à jour visuelle immédiate constitue un élément clé du caractère pédagogique du simulateur.

4.3.2 Registres 16 bits

Les registres X, Y, U, S et PC sont affichés sous forme hexadécimale sur 16 bits. Le compteur ordinal (PC) joue un rôle central, car il permet de visualiser l'avancement de l'exécution dans la mémoire programme.

4.3.3 Décomposition du registre de condition

Le registre CC est décomposé en huit drapeaux individuels. Chaque drapeau est affiché séparément, permettant de comprendre précisément l'impact des opérations arithmétiques et logiques sur l'état du processeur.

5 Modélisation de la mémoire

5.1 Rôle de la mémoire dans le simulateur

La mémoire constitue un élément central du simulateur MOF6809. Elle assure à la fois le stockage du programme assembleur (ROM) et des données manipulées lors de l'exécution (RAM).

La séparation logique entre ROM et RAM permet de refléter fidèlement l'architecture d'un système à microprocesseur classique.

5.2 Structure de données pour la ROM

La ROM est implémentée sous forme de tableau ou de structure indexée contenant des valeurs entières. Chaque valeur représente un octet de code machine simulé, issu de la phase de compilation du code assembleur.

Les adresses ROM sont manipulées sous forme hexadécimale, ce qui renforce la cohérence avec le langage assembleur 6809.

5.3 Structure de données pour la RAM

La RAM est également implémentée sous forme de tableau. Contrairement à la ROM, son contenu est modifiable dynamiquement au cours de l'exécution.

Les instructions de stockage et les opérations sur la pile entraînent des modifications directes de cette mémoire, lesquelles sont immédiatement reflétées dans l'interface graphique.

5.4 Mécanisme d'initialisation

Deux méthodes principales assurent l'initialisation de la mémoire :

— InitValeurROM, chargée de préparer la mémoire programme,

— InitValeurRAM, responsable de la mise à zéro ou de l'initialisation de la mémoire de données.

Ces méthodes sont invoquées lors du lancement du simulateur ou lors de la recompilation d'un nouveau programme assembleur.

5.5 Gestion de l'adressage hexadécimal

L'ensemble des adresses et valeurs mémoire est manipulé en hexadécimal. Des conversions explicites sont réalisées entre chaînes de caractères, valeurs entières et représentations binaires.

Cette gestion rigoureuse de l'adressage permet d'éviter les ambiguïtés et garantit une correspondance fidèle entre l'affichage graphique et les calculs internes du simulateur.

6 Analyse syntaxique et Parsing (Classe Instructions)

6.1 Rôle de l'analyse syntaxique dans MOF6809

Avant toute exécution, le simulateur MOF6809 doit transformer le texte assembleur saisi par l'utilisateur en une représentation exploitable par le moteur interne. Cette phase constitue une étape intermédiaire essentielle entre l'interface graphique et les mécanismes de décodage et de calcul.

La classe Instructions est responsable de cette analyse syntaxique. Elle agit comme un premier filtre, garantissant que le code assembleur respecte une structure minimale cohérente avant d'être interprété comme une suite d'instructions exécutables.

6.2 Lecture et nettoyage du code assembleur

Le code assembleur est fourni sous forme de texte brut via la zone de saisie de l'interface graphique. La classe Instructions procède à une lecture ligne par ligne de ce texte.

Les opérations suivantes sont réalisées :

— suppression des espaces inutiles,

— élimination des lignes vides,

— normalisation de la casse des mnémoniques.

Cette phase de nettoyage permet de réduire les ambiguïtés et de simplifier les étapes ultérieures d'analyse.

6.3 Séparation opérateur / opérande

Chaque ligne de code assembleur est ensuite analysée afin d'identifier :

— le mnémonique de l'instruction (opérateur),

— l'opérande éventuelle associée.

Les méthodes GetP1 et GetP2 assurent cette séparation. Le premier élément correspond au mnémonique (ex. LDA, ADDB), tandis que le second contient l'opérande sous forme de

chaîne de caractères.

Cette séparation est cruciale, car elle conditionne la détermination du mode d'adressage et la taille des données à manipuler.

6.4 Identification du mode d'adressage

La méthode `GetModeAdressage` analyse la syntaxe de l'opérande afin d'identifier le mode d'adressage utilisé. Dans le cadre du projet MOF6809, deux modes sont pris en charge :

— le mode immédiat, identifié par le symbole #,

— le mode inhérent, caractérisé par l'absence d'opérande.

Cette identification permet d'orienter le traitement vers les routines appropriées lors du décodage et de l'exécution.

6.5 Extraction des valeurs numériques

Les opérandes immédiats sont exprimés en hexadécimal. La méthode `GetValue` assure l'extraction et la conversion de ces valeurs sous forme entière.

Une attention particulière est portée à la distinction entre :

— valeurs sur 8 bits,

— valeurs sur 16 bits.

Cette distinction est essentielle afin de garantir la cohérence entre l'instruction, le registre ciblé et la taille des données manipulées.

7 Logique de décodage (Classe Decodage)

7.1 Fonction du décodage dans la chaîne d'exécution

Une fois l'analyse syntaxique réalisée, les instructions doivent être traduites en une forme compréhensible par le moteur de simulation. La classe Decodage joue le rôle de traducteur entre les mnémoniques assembleur et les actions internes du simulateur.

Elle constitue un lien fondamental entre la représentation textuelle du programme et son exécution effective.

7.2 Traduction mnémonique vers OpCode

Chaque mnémonique reconnu est associé à un code interne simulant l'OpCode du processeur Motorola 6809. Cette correspondance est implémentée sous forme de tables ou de structures conditionnelles.

Ce mécanisme permet de déterminer rapidement :

— le type d'opération à effectuer,

— la taille des données,

— les registres impliqués.

7.3 Traitement des instructions immédiates et inhérentes

Les instructions en mode immédiat nécessitent la prise en compte d'une valeur fournie explicitement dans le code. À l'inverse, les instructions inhérentes n'utilisent que l'état interne du processeur.

La classe Decodage distingue clairement ces deux cas et oriente le flux d'exécution vers les méthodes appropriées de l'unité arithmétique et logique.

7.4 Conversion hexadécimale et gestion du signe

Certaines instructions manipulent des valeurs signées. La méthode hexToSignedInt permet de convertir une valeur hexadécimale en entier signé, en tenant compte de la taille

(8 ou 16 bits).

Cette conversion est indispensable pour simuler correctement les opérations arithmétiques et les effets sur les drapeaux du registre CC.

7.5 Table de correspondance des instructions

L'ensemble du décodage repose sur une table de correspondance entre :

— les mnémoniques assembleur,

— les routines internes à appeler,

— les effets attendus sur les registres et la mémoire.

Cette table constitue une abstraction du jeu d'instructions réel du 6809, adaptée au périmètre fonctionnel du projet MOF6809.

8 Unité Arithmétique et Logique (Classe UAL)

[Image of Arithmetic Logic Unit diagram with flags and registers]

8.1 Architecture générale de la classe UAL

La classe UAL représente le cœur computationnel du simulateur MOF6809. Elle centralise l'ensemble des opérations arithmétiques, logiques et de transfert, ainsi que la mise à jour du registre de condition.

Toutes les instructions décodées convergent vers cette classe, ce qui en fait le composant le plus critique du projet.

8.2 Gestion des drapeaux du registre CC

Le registre de condition (CC) est mis à jour après chaque opération pertinente. La classe UAL gère explicitement les drapeaux suivants :

8.2.1 Retenue (Carry – C)

Le drapeau C est positionné lorsqu'une opération génère une retenue hors de la capacité du registre cible.

8.2.2 Zéro (Z) et Signe (N)

Le drapeau Z indique un résultat nul, tandis que le drapeau N reflète le bit de signe du résultat.

8.2.3 Débordement (Overflow – V) et Demi-retenue (H)

Le drapeau V signale un dépassement de capacité en arithmétique signée. Le drapeau H est utilisé dans certains calculs intermédiaires, notamment pour simuler fidèlement le comportement du processeur réel.

8.3 Algorithmes des opérations arithmétiques

Les instructions telles que ADDA, SUBD et MUL sont implémentées à l'aide d'algorithmes manipulant explicitement des représentations binaires.

Chaque opération suit une séquence précise :

— conversion des opérandes,
— calcul du résultat,
— mise à jour des registres,
— mise à jour des drapeaux.

8.4 Opérations logiques

Les opérations logiques (ANDA, ORB, EOR) sont implémentées à l'aide d'opérations bit à bit. Elles illustrent de manière claire l'impact direct des manipulations binaires sur les registres et le registre de condition.

8.5 Instructions de transfert et d'échange

Les instructions TFR et EXG permettent respectivement le transfert et l'échange de valeurs entre registres. Leur implémentation nécessite une gestion rigoureuse des tailles de données et des correspondances entre registres source et destination.

8.6 Implémentation des instructions de pile

Les instructions PSHS et PSHU assurent la sauvegarde de l'état du processeur sur la pile. La classe UAL interagit directement avec la mémoire RAM pour empiler les registres spécifiés, en respectant l'ordre et les tailles définies par l'architecture 6809.

Ces instructions constituent un élément clé pour illustrer la gestion du contexte processeur et la notion de pile dans un système informatique.

9 Gestion des erreurs et robustesse (Classe Erreurs)

9.1 Rôle de la gestion des erreurs dans MOF6809

Dans un contexte pédagogique, la gestion des erreurs joue un rôle central. Elle permet non seulement d'éviter des comportements imprévus du simulateur, mais également d'accompagner l'utilisateur dans l'apprentissage du langage assembleur Motorola 6809.

La classe `Erreurs` constitue le mécanisme principal de validation et de diagnostic du programme assembleur fourni par l'utilisateur. Elle intervient avant toute phase d'exécution, garantissant que seules des instructions syntaxiquement et sémantiquement correctes sont traitées par le moteur de simulation.

9.2 Stratégie de validation pré-compilation

Avant le décodage et l'exécution, chaque ligne du code assembleur est analysée afin de vérifier :

— l'existence du mnémonique,
— la validité du nombre d'opérandes,
— la cohérence entre l'instruction et le mode d'adressage utilisé.

Cette validation préliminaire permet d'éliminer un grand nombre d'erreurs courantes dès les premières étapes du traitement.

9.3 Vérification de la syntaxe et des opérandes

La classe `Erreurs` contrôle la conformité de la syntaxe des opérandes, notamment :

— la présence correcte du symbole # pour le mode immédiat,
— le format hexadécimal des constantes,
— l'absence d'opérandes pour les instructions inhérentes.

Ces vérifications permettent de prévenir des incohérences susceptibles de compromettre l'exécution correcte du programme.

9.4 Contrôle de cohérence 8 bits / 16 bits

Certaines instructions opèrent sur des registres 8 bits, tandis que d'autres manipulent des registres 16 bits. La classe `Erreurs` vérifie que la taille de la valeur fournie est compatible avec le registre cible.

Ce mécanisme empêche par exemple le chargement d'une valeur 16 bits dans un registre 8 bits, erreur fréquente chez les débutants.

9.5 Rapport d'erreurs à l'utilisateur

Les erreurs détectées sont communiquées à l'utilisateur via la méthode `AfficheCompilation`. Les messages sont affichés dans une zone dédiée de l'interface graphique, sous une forme claire et explicite.

Cette approche favorise un apprentissage progressif, en guidant l'utilisateur vers la correction de ses erreurs plutôt qu'en bloquant brutalement l'exécution.

10 Flux d'exécution et simulation

10.1 Cycle de vie d'une instruction dans MOF6809

Dans le simulateur MOF6809, chaque instruction suit un cycle de vie structuré :

1. saisie dans l'interface graphique,
2. analyse syntaxique,
3. validation et gestion des erreurs,
4. décodage,
5. exécution par l'UAL,
6. mise à jour des registres et de la mémoire,
7. rafraîchissement de l'interface graphique.

Ce cycle reproduit de manière abstraite le fonctionnement interne d'un processeur réel.

10.2 Processus de compilation

La phase dite de « compilation » consiste à transformer le texte assembleur en structures internes exploitables. Les instructions sont converties en valeurs stockées dans la ROM simulée, tandis que les opérandes sont préparés pour l'exécution.

Cette phase établit un lien clair entre le programme textuel et sa représentation mémoire.

10.3 Exécution pas-à-pas

Le mode pas-à-pas constitue l'un des aspects pédagogiques majeurs du projet. À chaque étape :

— une seule instruction est exécutée,
— les registres sont mis à jour,
— la mémoire est modifiée si nécessaire,
— l'interface graphique est synchronisée avec l'état interne.

Ce mécanisme permet à l'utilisateur d'observer précisément l'impact de chaque instruction sur le processeur simulé.

10.3.1 Historisation des états

Des structures de type ArrayList sont utilisées pour conserver l'historique des états successifs. Cette historisation facilite le suivi de l'évolution du programme et renforce la compréhension du flux d'exécution.

10.4 Exécution globale

En complément du mode pas-à-pas, le simulateur propose une exécution globale du programme. Dans ce mode, l'ensemble des instructions est exécuté séquentiellement sans interruption, tout en conservant la mise à jour visuelle finale de l'état du processeur.

11 Bilan, limites et perspectives

11.1 Synthèse des réalisations

Le projet MOF6809 a permis de concevoir un simulateur fonctionnel du microprocesseur Motorola 6809. Il intègre :

— une interface graphique pédagogique,

— un moteur de simulation structuré,

— une gestion cohérente des registres, de la mémoire et des drapeaux.

L'ensemble constitue une base solide pour l'étude pratique de l'architecture des processeurs.

11.2 Analyse critique et contraintes temporelles

Le développement du projet s'est déroulé dans un cadre temporel limité. Cette contrainte a influencé certains choix, notamment la restriction volontaire du nombre de modes d'adressage implémentés.

Toutefois, cette limitation a permis de concentrer les efforts sur la fiabilité, la clarté du code et la valeur pédagogique du simulateur.

11.3 Fonctionnalités non implémentées

Certains modes d'adressage du Motorola 6809 n'ont pas été intégrés, notamment :

— le mode indexé,

— le mode relatif,

— l'adressage indirect.

Leur implémentation nécessiterait une extension du moteur de décodage et de la gestion de la mémoire.

11.4 Perspectives d'amélioration

Plusieurs axes d'évolution peuvent être envisagés :

— enrichissement du jeu d'instructions,
— optimisation du code de l'UAL,
— amélioration de l'ergonomie de l'interface graphique,
— ajout de mécanismes de débogage avancés.

12 Conclusion

Le projet MOF6809 constitue une réalisation académique complète et cohérente, illustrant de manière concrète les principes fondamentaux de l'architecture des ordinateurs.

À travers la simulation du microprocesseur Motorola 6809, ce projet a permis de relier théorie et pratique, tout en mettant en évidence les interactions complexes entre analyse syntaxique, décodage, calcul arithmétique et gestion de la mémoire.

Malgré les contraintes temporelles ayant limité l'implémentation exhaustive de tous les modes d'adressage, le simulateur remplit pleinement ses objectifs pédagogiques. Il offre une base évolutive, susceptible d'être enrichie dans le cadre de travaux futurs ou de projets académiques plus avancés.