LE RAPPORT DU PROJET

SIMULATEUR DU MICROPROCESSEUR MOTOROLA 6809

Encadré par : Mr. BENALLA Hicham

ANNÉE UNIVERSITAIRE : 2025-2026

REMERCIEMENTS

Nous tenons à exprimer notre gratitude a l'établissement université HASSAN 1ER (Faculté des sciences et techniques) pour avoir mis à disposition les ressources qui ont permis la concrétisation de ce projet d'architecture
du microprocesseur Motorola 6809.

Notre gratitude va en premier lieu à Mr BENALLA Hicham , notre enseignant en architecture des ordinateurs, pour ses explications claires sur le microprocesseur 6809, sa disponibilité et son expertise qui nous ont guidés dans la compréhension des spécificités du 6809, de son jeu d'instructions et de ses modes d'adressage avancés et pour nous donner la chance de travailler en groupe et présenter notre projet devant vous.

Enfin, nous remercions tout particulièrement les personnes qui ont contribué à la réalisation de ce projet.

PLAN
DU TRAVAIL

1 Introduction

2 Membres du groupe

3 Présentation du Microprocesseur Motorola 6809

4 Test du programme

5 Résultats obtenus

6 Conclusion

1 INTRODUCTION

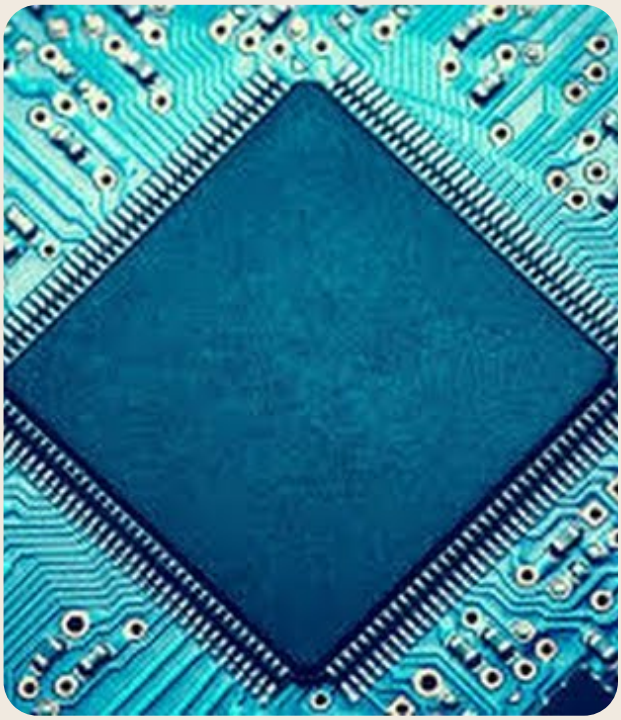
MON RAPPORT DE PROJET

Dans ce projet, On présentera un simulateur du microprocesseur Motorola 6809 qui va nous aider de mieux comprendre l'architecture des ordinateurs.

Ce projet nous a permis de comprendre le comportement, les bases et le fonctionnement du microprocesseur à partir de ses instructions et ses modes d'adressages qu'on va les expliquer en détails dans la suite de notre présentation .

Objectifs

Avec ce simulateur, on peut exécuter des programmes écrits pour le 6809, suivre l'état du mémoire et des registres en direct et comprendre comment les instructions sont traitées.



2 PRÉSENTATION DES MEMBRES DU GROUPE

L'ÉQUIPE:

CHAFAOUI AMAL
23012031

CHAOUKI HAJAR
23012038

LAHRACH CHAIMAE
23011785

TALEB NEZHA
23011985

3 PRÉSENTATION DU MICROPROCESSEUR MOTOROLA 6809

1.LA CLASSE REGISTER :

Dans la classe Register, on a declaré les registres A, B, X, Y, PC, SP, CC, u, s, dp ainsi que flags de 8 bits de condition (C, V, Z, N, I, H, F, E) qui sont utilisés pour stocker les résultats des opérations arithmétiques et logiques, chaque flag occupe un bit spécifique dans le registre CC.

Les registres A et B ce sont des registres de 8 bits qui servent d'accumulateurs, les registres X et Y sont des registres de 16 bits qui servent d'index, le registre PC est le compteur de programme qui contient l'adresse de la prochaine instruction à exécuter, le registre SP est le pointeur de pile qui contient l'adresse de la pile.

Les flags de condition sont utilisés pour indiquer les résultats des opérations, tels que:

-Débordement (Overflow OV ou V)
-Nullité (Zéro Z)
-Négativité (Sign S)
-Retenue (Carry C)
-Retenue intérmidiaire (Auxillairy-Carry H)
-Parité (Parity P)
-Etat de sauvegarde (E)
-Masque des interruptions (IRQ I)
-Masque des interruptions (FIRQ F)

Nous avons également implémenté les méthodes get() et set() pour accéder et modifier ces registres et flags, les méthodes update() pour mettre à jour les flags en fonction des résultats des opérations, la méthode reset() pour réinitialiser les registres et les flags et finalement la méthode printState() pour afficher l'état des registres et des flags.

3 PRÉSENTATION DU MICROPROCESSEUR MOTOROLA 6809

2.LES CLASSES RAM,ROM ET MEMOIRE

RAM:

On a presenté une classe RAM dans notre projet comme un tableau de 1024 octets qui représentera la mémoire RAM du notre simulateur. Chaque élément du tableau représente une adresse mémoire du 0000 à 03FF .

Elle contient deux méthodes :

·read(adresse) :lire une valeur à une adresse dans la mémoire RAM.

·write(adresse,valeur) :écrire une valeur à une adresse dans la mémoire RAM.

ROM:

Il prend un tableau aussi de 1024 octets de FC00 à FFFF qui représente le programme à charger dans la mémoire ROM.

Elle contient une seule méthode :

·read(adresse) : lire une valeur à une adresse dans la mémoire ROM.

MEMOIRE:

La mémoire est représentée par la classe memoire. Elle fait l'appel à RAM et ROM.

Elle contient deux méthodes :

·read(adresse) : lire une valeur à une adresse dans la mémoire mémoire. Elle vérifie si l'adresse se trouve dans l'intervalle de la mémoire RAM ou ROM et fait l'appel à la méthode read correspondante.

·write(adresse,valeur) : écrire une valeur à une adresse dans la mémoire. Elle vérifie si l'adresse se trouve dans l'intervalle de la mémoire RAM et fait l'appel à la méthode write. Si l'adresse est dans l'intervalle de la mémoire ROM, elle ne fait pas l'appel car la mémoire ROM est en lecture seulement.

3 PRÉSENTATION DU MICROPROCESSEUR MOTOROLA 6809

3.LA CLASSE INSTRUCTION

Cette classe implémente le cœur de l'assembleur du simulateur, convertissant les instructions assembleur en code machine exécutable.

Fonctionnalités principales :

- Analyse syntaxique : Parse les instructions du type LDA #$10 ou JMP $1234
- Détection des modes d'adressage : Reconnaît 9 modes (immédiat, direct, étendu, indexé, relatif, inherent, etc.)
- Génération de code machine : Produit les octets binaires correspondants
- Validation : Vérifie la syntaxe et fournit des messages d'erreur détaillés

Structure clé :

- Champs final pour opcode et operand (immuables après construction)
- Méthode detecteMode() : analyse l'opérande pour déterminer le mode d'adressage
- Méthode getOpcodeHex() : table de conversion opcode → hexadécimal selon le mode
- Méthode assemble() : génère le tableau d'octets final (opcode + opérande formaté)

Modes d'adressage supportés :

- Immédiat : LDA #$FF
- Direct : STA $10 (adresse 8 bits)
- Étendu : JMP $1234 (adresse 16 bits)
- Indexé : LDA 5,X ou LDA ,Y
- Relatif : BEQ $FC (pour les branchements)
- Inherent : RTS, NOP (pas d'opérande)

I**nstructions implémentées dans notre projet :**

- Transfert : LD (Load), ST (Store)
- Arithmétique : ADD, SUB, INC, DEC
- Logique : AND, OR
- Comparaison : CMP
- Contrôle : JMP, BNE, BEQ

Cette classe assure la cohérence entre la syntaxe assembleur lisible et la représentation binaire exécutable par le simulateur 6809.

3 PRÉSENTATION DU MICROPROCESSEUR MOTOROLA 6809

4.LA CLASSE CPU

La classe Cpu représente le processeur Motorola 6809 dans notre émulateur. Elle gère:
-la lecture des instructions.
-leur exécution.
-la manipulation des registres et de la mémoire.

La classe contient des méthodes pour initialiser,exécuter et contrôler l'état du CPU,
ainsi que des fonctions auxiliaires pour les opérations arithmétiques et logiques.

1. **Attributs principaux**
-reg : une instance de la classe Register qui contient tous les registres du CPU (A, B, X, Y, PC, SP, DP, et les flags conditionnels).
-mem : une instance de la classe Memoire représentant la mémoire du CPU.
-halted : un booléen indiquant si le CPU est arrêté (true) ou prêt à exécuter (false).
Ces attributs permettent au CPU de suivre l'état interne et de savoir quelles instructions exécuter.

2. **Initialisation et contrôle**
reset() : remet le CPU à l'état initial. Il réactive le CPU (halted = false) et réinitialise tous les registres et flags via reg.reset().
isHalted() : retourne l'état du CPU, permettant de vérifier s'il est arrêté ou non.
Ces méthodes sont essentielles pour démarrer ou redémarrer l'exécution d'un programme.

3. **Lecture des instructions**
fetch8() : lit un octet depuis la mémoire à l'adresse pointée par le compteur de programme (PC), puis incrémente le PC.
fetch16() : lit deux octets consécutifs (format big-endian) et les combine en un entier 16 bits. Cette opération est utilisée pour les instructions nécessitant une adresse ou une valeur sur deux octets.
Ces méthodes permettent de décoder correctement les instructions et leurs opérandes.

4. **Exécution des instructions**
-step() : exécute une seule instruction pointée par le PC. Elle lit l'opcode, puis effectue le traitement correspondant via un switch sur l'opcode.

Les instructions couvertes incluent :
Chargement / stockage : LDA, LDB, LDX, LDY, STA, STB, STX (avec modes direct, étendu et indexé).
Opérations arithmétiques : ADD, SUB, MUL, INCA, DECA.
Opérations logiques : ANDA, ANDB, COMA, COMB, CLRA, CLRB.
Comparaisons : CMP A, CMP B.
Branches et sauts : JMP, BRA, BNE, BEQ, BMI, BPL.
Arrêt du CPU : HALT.
-run() : exécute le programme jusqu'à ce que le CPU soit arrêté (halted = true), en appelant step() de manière répétée.

3 PRÉSENTATION DU MICROPROCESSEUR MOTOROLA 6809

4.LA CLASSE CPU

5. **Méthodes auxiliaires**
Pour simplifier le traitement des instructions, la classe contient des méthodes privées telles que :
-Opérations arithmétiques : addA(), addB(), subA(), subB(). Elles mettent également à jour les flags conditionnels (N, Z, H, C).
-Comparaisons : cmpA(), cmpB(). Elles ne modifient pas les registres mais mettent à jour les flags pour les instructions de branchement.
-Branches conditionnelles : branchIf(boolean condition) permet de sauter à une adresse relative en fonction d'une condition (utile pour BNE, BEQ, BMI, etc.).

6**. Gestion de la mémoire et des registres**
La classe utilise les registres et flags via reg pour :
Calculer les adresses mémoire (par exemple pour le mode indexé).
Mettre à jour les flags conditionnels après chaque opération.
Lire et écrire des valeurs dans la mémoire via mem.write() et mem.read().

5.LA CLASSE INTERFACE

Le code présenté crée une interface graphique complète pour un simulateur du microprocesseur Motorola 6809, entièrement développé en Java. Il utilise deux bibliothèques graphiques de base du langage : AWT (Abstract Window Toolkit) et Swing. AWT fournit les classes nécessaires pour gérer les fenêtres, les événements et le dessin graphique de base, notamment via les packages java.awt et java.awt.event. Swing, qui fait partie de javax.swing, enrichit cette base avec des composants plus modernes, flexibles et personnalisables, comme JFrame, JTextArea, JButton et JToolBar. Cette combinaison permet de créer une interface réactive et modulaire, tout en ajoutant des éléments graphiques personnalisés, comme les lignes qui relient les registres à l'UAL. Ces lignes sont dessinées à l'aide de la méthode paintComponent héritée de la classe JPanel, qui utilise le canevas graphique d'AWT via Graphics. L'ensemble est conçu selon un modèle d'événements strict, garantissant une interaction fluide entre l'utilisateur, l'éditeur de code, le moteur d'assemblage et le simulateur du processeur.

3 PRÉSENTATION DU MICROPROCESSEUR MOTOROLA 6809

L'architecture générale est basée sur une séparation claire entre la logique de simulation (classes Cpu, Register, Memoire, RAM, ROM, Instruction) et l'interface utilisateur. La méthode main sert de point d'entrée et initialise d'abord les composants essentiels de la simulation. Cela inclut une instance de mémoire avec une ROM de 1024 octets, qui est vide au départ, une RAM conforme à l'architecture cible, et un registre et un processeur associés. Le processeur est immédiatement réinitialisé via la méthode reset() pour établir un état de départ connu. Cette initialisation garantit que chaque session de simulation commence dans des conditions identiques.

L'interface principale se compose de plusieurs fenêtres indépendantes mais synchronisées. La fenêtre principale affiche une barre de menus complète (Fichier, Simulation, Outils, Fenêtres, Options, Aide) et une barre d'outils graphique intuitive. Cette barre d'outils offre des raccourcis visuels pour des opérations courantes : ouverture et enregistrement de fichiers, réinitialisation du processeur, déclenchement d'interruptions (IRQ, FIRQ, NMI), exécution continue, arrêt et exécution pas à pas. L'exécution continue s'effectue dans un thread dédié pour ne pas bloquer l'interface graphique. Les mises à jour de l'affichage des registres et de la mémoire se font de manière thread-safe grâce à SwingUtilities.invokeLater. Cela assure une utilisation fluide même lors de longues boucles d'exécution.

La fenêtre des registres représente visuellement l'architecture interne du 6809. Elle affiche les huit registres principaux : PC (compteur de programme), A et B (accumulateurs 8 bits), X et Y (registres d'index 16 bits), S et U (pointeurs de pile système et utilisateur), et DP (registre de page directe 8 bits). De plus, les huit bits d'état du registre de flags (H, N, Z, V, C, F, I, E) sont affichés en binaire. Une zone graphique centrale représente l'Unité Arithmétique et Logique (UAL) et montre visuellement les liens entre les registres A, B et DP. Ces liens sont tracés dynamiquement à l’aide d’une surcharge de paintComponent dans un JPanel personnalisé. Cette approche utilise efficacement AWT pour le dessin et Swing pour la gestion des conteneurs. Cette représentation aide à comprendre le flux de données dans le processeur..

3 PRÉSENTATION DU MICROPROCESSEUR MOTOROLA 6809

L'éditeur de code source est le cœur de l'interaction utilisateur. Il permet de saisir du code assembleur dans une zone de texte multiligne (JTextArea), avec une police appropriée et une mise en page claire. Un bouton déclenche la compilation du code via la méthode assemblerCode(). Cette méthode analyse le contenu ligne par ligne, ignore les commentaires (qui commencent par ;) et les lignes vides, et interprète chaque instruction à l'aide d'une classe Instruction, qui n'est pas incluse ici mais qui est supposée être bien implémentée. En cas d'erreur syntaxique ou de dépassement de la capacité mémoire (1024 octets), un message d'erreur apparaît via JOptionPane. À la fin d'un assemblage réussi, la nouvelle image binaire est chargée dans la ROM, le processeur est réinitialisé, et toutes les fenêtres d'affichage sont mises à jour automatiquement. Cette boucle de rétroaction immédiate est essentielle pour un outil d'apprentissage efficace.

Enfin, deux fenêtres dédiées affichent le contenu de la RAM (adresses 0x0000 à 0x03FF) et de la ROM (adresses 0xFC00 à 0xFFFF). Pour améliorer la lisibilité, seules les cases mémoire dont la valeur diffère de 0xFF (valeur par défaut non initialisée) sont montrées. Cela évite l'encombrement visuel et se concentre sur les données importantes modifiées par l'exécution du programme. Cette approche est particulièrement utile lors de démonstrations de programmes courts ou de débogage d'algorithmes simples.

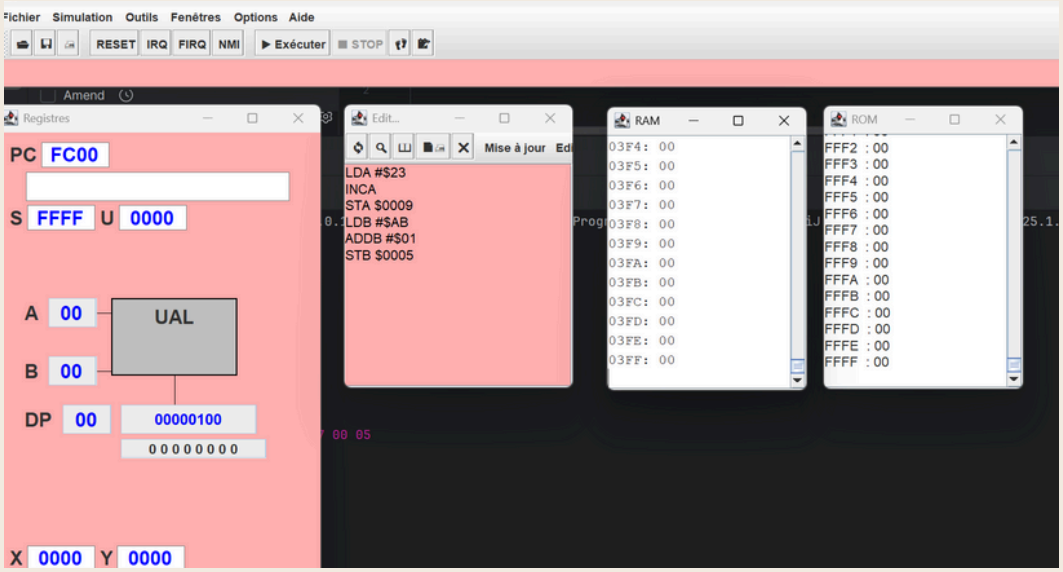
En somme, cette interface graphique offre un environnement de simulation complet, rigoureux et éducatif, parfaitement adapté à l'enseignement de l'architecture des microprocesseurs et de la programmation en assembleur. Elle combine une présentation claire des concepts fondamentaux (registres, mémoire, exécution, interruptions) avec des outils pratiques (édition, assemblage, visualisation dynamique), rendant l'apprentissage du Motorola 6809 accessible et engageant. L'utilisation habile d'AWT pour le dessin de bas niveau et de Swing pour les composants interactifs montre une bonne maîtrise des outils graphiques de Java dans un contexte éducatif exigeant

4 TEST DU PROGRAMME

LDA #$23
Charge la valeur hexadécimale 23 (35 en décimal) dans le registre A
  Mode immédiat.
INCA
Incrémente le registre A de 1
  A = 23 + 1 = 24 (36 en décimal).
STA $0009
Stocke le contenu du registre A à l'adresse mémoire 0009
  Mode étendu.
LDB #$AB
Charge la valeur hexadécimale AB (171 en décimal) dans le registre B
  Mode immédiat.
ADDB #$01
Additionne la valeur 01 au registre B
  B = AB + 01 = AC (172 en décimal).
STB $0005
Stocke le contenu du registre B à l'adresse mémoire 0005.

État actuel de la simulation :

- Program Counter (PC) : FC00
- Adresse de début du programme.
- Registres :
  - A = 00
  - B = 00
  - Le programme n'a pas encore été exécuté.
- Mémoire RAM :
  - Toutes les valeurs sont à 00
  - Aucune écriture mémoire effectuée.
- Mémoire ROM :
  - Contient le programme assemblé.

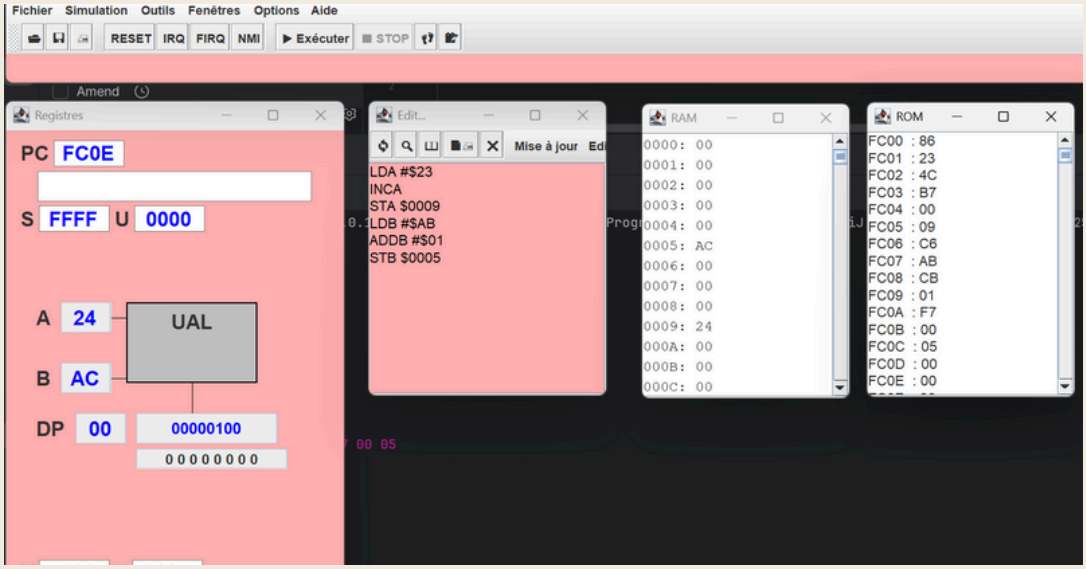


4 RESULTATS OBTENUS

- **Explication étape par étape :**

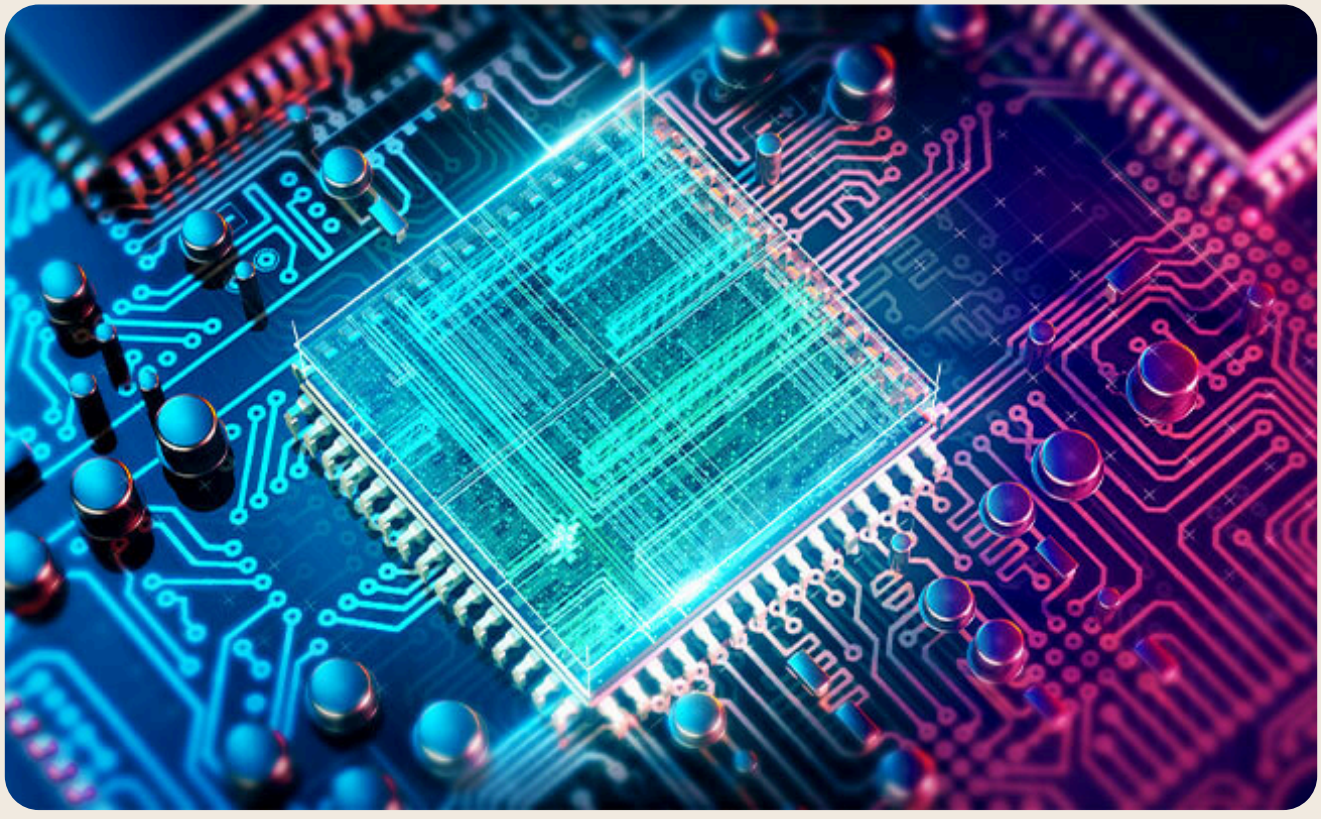
1. LDA #$23
2. Charge la valeur 0x23 (35 en décimal) dans le registre A
3. INCA
4. Incrémente A
5. A devient 0x24
6. STA $0009
7. Stocke la valeur de A (0x24) en mémoire à l'adresse $0009
8. LDB #$AB
9. Charge la valeur 0xAB dans le registre B
10. ADDB #$01
11. Ajoute 1 à B
12. B devient 0xAC
13. STB $0005
14. Stocke la valeur de B (0xAC) en mémoire à l'adresse $0005

- **Résultat final :**
- Registre A = 0x24
- Registre B = 0xAC
- RAM[0009] = 0x24
- RAM[0005] = 0xAC



6 CONCLUSION

Ce projet nous a fait découvrir comment fonctionne vraiment le processeur Motorola 6809. En créant les classes **Motorola** et **CPU**, on a compris comment un processeur lit les instructions, utilise ses registres et exécute chaque opération étape par étape.
Grâce à la programmation orientée objet, on a organisé notre code de façon claire : chaque partie du simulateur a un rôle précis, comme dans un vrai ordinateur. Cela nous a aidés à mieux comprendre comment tous les éléments travaillent ensemble.
Ce projet nous a aussi appris à travailler en équipe et à résoudre des problèmes techniques complexes. C'est une bonne base pour apprendre d'autres choses en informatique, et ça montre qu'avec de la patience et de l'organisation, on peut recréer le fonctionnement d'un vrai processeur en logiciel.



RAPPORT
DE PROJET

SIMULATEUR DU MICROPROCESSEUR MOTOROLA 6809

MERCI