كلية العلوم والتقنيات سطات
FACULTÉ DES SCIENCES ET TECNHIQUES SETTAT

PRÉSENTATION DU MICROPROCESSEUR MOTOROLA 6809 SIMULATEUR

Réaliser et présenter par :

BENNANI MOHAMED

BENCHAKROUN KRIMI SAAD

OBJECTIF DU PROJET

L’objectif de ce projet est de concevoir et réaliser une simulation logicielle du microprocesseur Motorola 6809. Il vise à comprendre l’architecture interne du microprocesseur, notamment les registres, la mémoire et les modes d’adressage.

Le projet permet de relier les notions théoriques du cours à une implémentation concrète en langage Java. Enfin, il constitue un outil pédagogique facilitant l’apprentissage de l’architecture des ordinateurs et de la programmation bas niveau

DÉFINITION D’UN MICROPROCESSEUR

Un microprocesseur est un circuit intégré complexe chargé d’interpréter et d’exécuter les instructions d’un programme.

Il assure le traitement des données grâce à l’unité arithmétique et logique (UAL) et le contrôle du système par l’unité de commande.

Il communique avec la mémoire et les périphériques à travers des bus de données, d’adresses et de contrôle. Le microprocesseur constitue ainsi le cœur de fonctionnement d’un ordinateur.

Le microprocesseur 6809

Le microprocesseur Motorola 6809 est un processeur qui traite des données sur 8 bits, mais dont l’architecture interne permet également des opérations sur 16 bits. Il est réalisé en technologie MOS, utilisée pour la fabrication des circuits intégrés.

Ce microprocesseur est conditionné dans un boîtier comportant 40 broches, permettant les connexions avec la mémoire et les périphériques.

Les registres du microprocesseur 6809

Le microprocesseur 6809 possède 9 registres programmables :

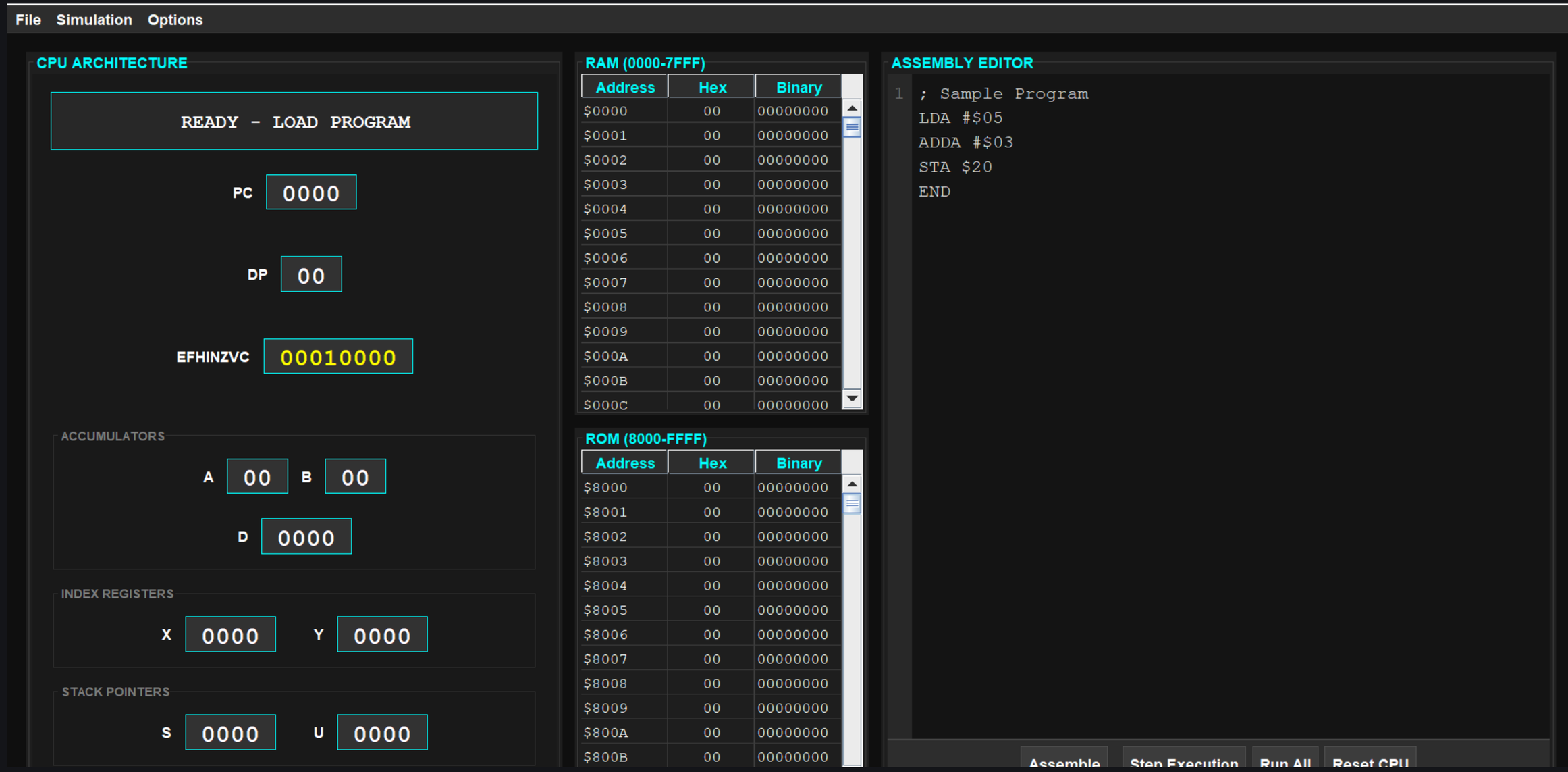
- Accumulateurs : A, B, D (virtuelle)
- Registres d’index : X, Y
- Pointeurs de pile : S, U
- Registre de page : DP
- Compteur ordinal : PC
- Registre d’état : CCR

Les modes d’adressages

Le microprocesseur 6809 possède 9 modes d’adressage :

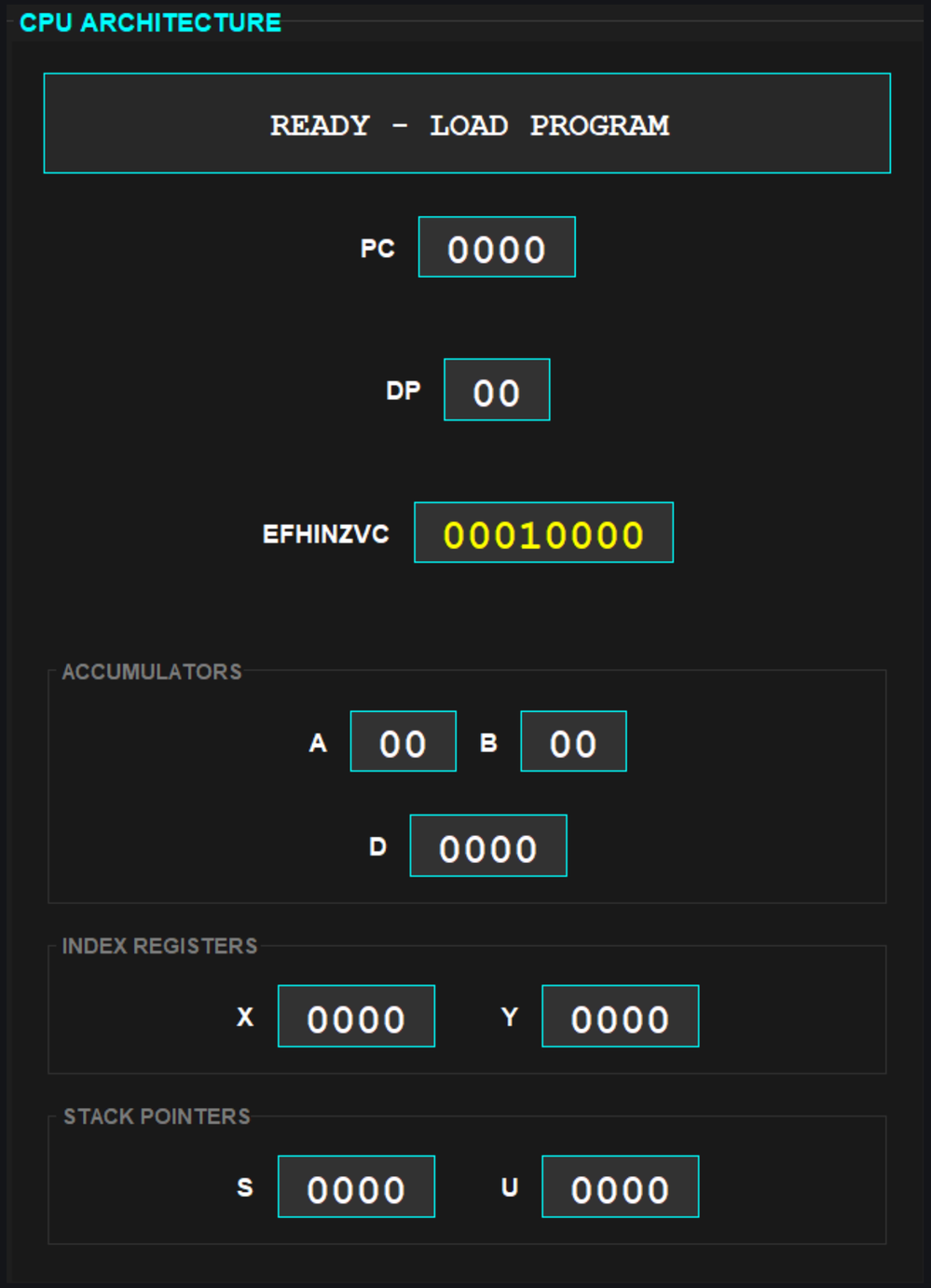
- **l’adressage inhérent**
- **l’adressage immédiat**
- **l’adressage étendu (direct)**
- **l’adressage étendu indirect**
- **l’adressage direct**
- **l’adressage par registre**
- **l’adressage indexé direct**
- **l’adressage indexé indirect**
- **l’adressage relatif**

Interface Programme



CPU

Les registres du CPU affichent les valeurs courantes (PC, A, B, X, Y, S, U, CCR, DP).



RAM

RAM (0000-7FFF)

Address	Hex	Binary
$0000	86	10000110
$0001	05	00000101
$0002	8B	10001011
$0003	03	00000011
$0004	97	10010111
$0005	20	00100000
$0006	00	00000000
$0007	00	00000000
$0008	00	00000000
$0009	00	00000000
$000A	00	00000000
$000B	00	00000000
$000C	00	00000000

La RAM (mémoire vive) sert à stocker temporairement les données et instructions pendant l’exécution du programme.

ROM

ROM (8000-FFFF)

Address	Hex	Binary
$8000	00	00000000
$8001	00	00000000
$8002	00	00000000
$8003	00	00000000
$8004	00	00000000
$8005	00	00000000
$8006	00	00000000
$8007	00	00000000
$8008	00	00000000
$8009	00	00000000
$800A	00	00000000
$800B	00	00000000
$800C	00	00000000

La ROM contient le programme et les vecteurs système : elle est en lecture seule et fournit au CPU les instructions à exécuter.

Éditeur Assembleur

RAM (0000-7FFF)

Address	Hex	Binary
$0000	00	00000000
$0001	00	00000000
$0002	00	00000000
$0003	00	00000000
$0004	00	00000000
$0005	00	00000000
$0006	00	00000000
$0007	00	00000000
$0008	00	00000000
$0009	00	00000000
$000A	00	00000000
$000B	00	00000000
$000C	00	00000000

L’éditeur assembleur est l’interface où l’on écrit le code source en assembleur, puis on peut l’assembler, le charger en mémoire et l’exécuter pas à pas ou en continu.

Résumé des Classes et Codes

Chaque classe joue un rôle spécifique :

- CPU.java incarne le cœur du processeur, avec ses registres et ses instructions.
- Memory.java, RAM.java et ROM.java gèrent l'organisation et l'accès aux données en mémoire.
- Assembler.java traduit le langage assembleur en langage machine.
- Instruction.java représente chaque instruction comme une entité autonome.
- Simulator.java coordonne l'ensemble, reliant CPU, mémoire et assembleur.
- Enfin, les classes de l'interface graphique (ViewCPU, ViewMemory, ViewEditor, Main, CustomMenuBar) permettent à l'utilisateur d'interagir avec le simulateur de manière intuitive et visuelle.

CPU.java

- Rôle : Implémente la logique du microprocesseur 6809.
- Contenu principal :
- Registres : A, B, D, X, Y, S, U, PC, DP, CC.
- Cycle d’exécution : fetch → decode → execute → update.
- Modes d’adressage : immédiat, direct, étendu, relatif...
- Gestion des drapeaux (Z, N, C, V).

Memory.java / RAM.java / ROM.java

- Rôle : Gérer la mémoire (lecture/écriture).
- Organisation :
- RAM : 32KB (0x0000–0x7FFF).
- ROM : 32KB (0x8000–0xFFFF).
- Vecteur de reset à 0xFFFE–0xFFFF.
- Code clé : read() et write() qui distinguent RAM et ROM.

Assembler.java

- Rôle : Convertir le code assembleur en machine code.
- Étapes :

Découpage : découpe des lignes.

Analyse syntaxique: identification du mnémonique (nom de l’instruction) et du mode d’adressage.

Génération du code machine : attribution des opcodes et calcul des offsets (branches).

Chargement en mémoire : chargement du programme en mémoire.

Instruction.java

- Rôle : Représenter une instruction assembleur.
- Attributs :

mnemonic (ex. LDA, STA).
addressMode (IMMEDIATE, DIRECT, etc.).
operand (valeur/adresse).
opcode (byte machine).
size (nombre d’octets).

Simulator.java

- Rôle : Coordonne CPU, mémoire et assembleur.
- Fonctions :

Charger un programme.

Exécuter pas-à-pas ou en continu.

Gérer l’état global du simulateur.

Interface graphique (Java Swing)

- Rôle : Permet une visualisation claire et interactive (mode sombre, couleurs, numéro de ligne)

Main.java : point d’entrée, création de la fenêtre.

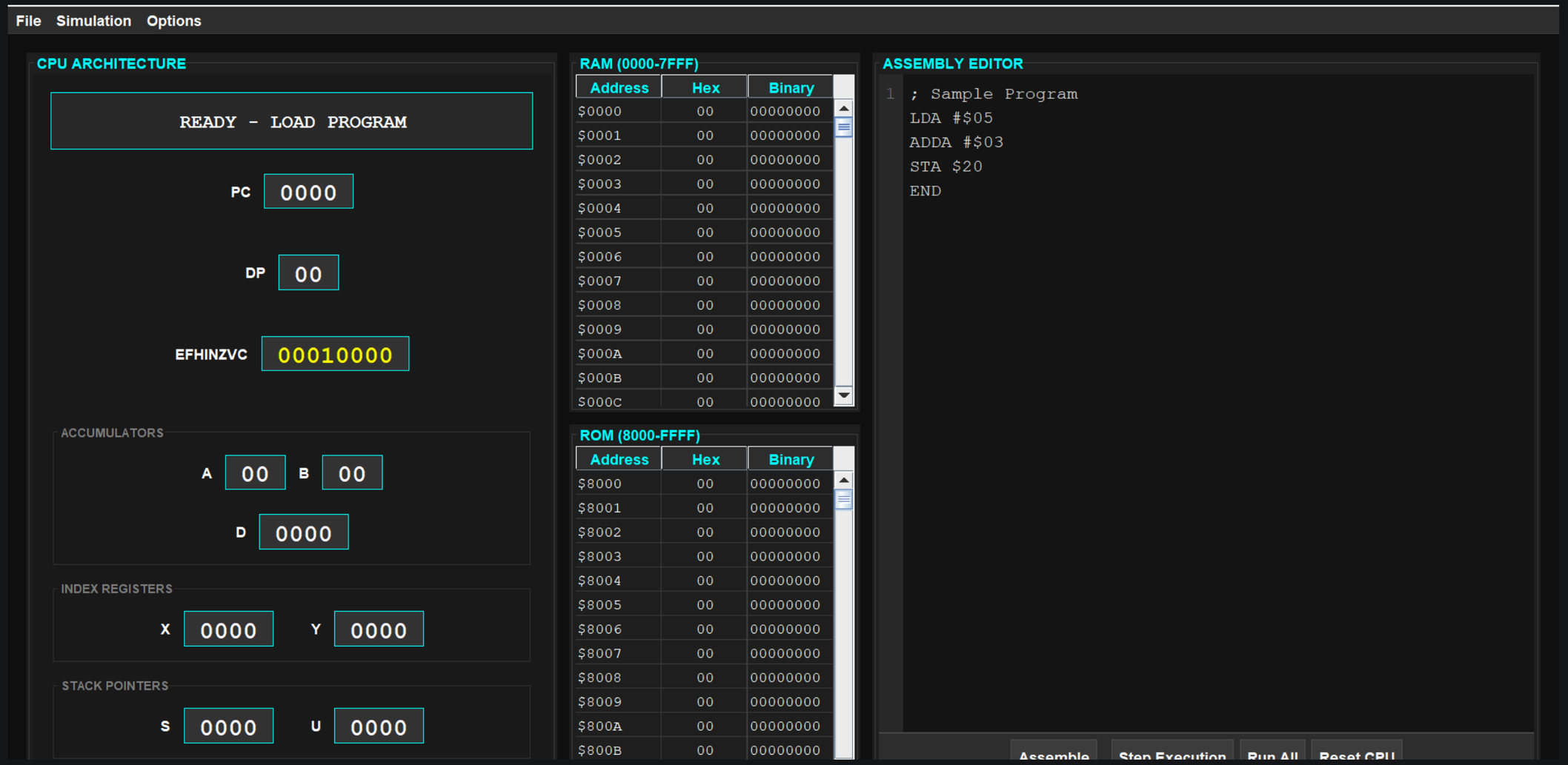
ViewCPU.java : affiche les registres, flags et instruction courante.

ViewMemory.java : tableau hexadécimal/binaire de la mémoire.

ViewEditor.java : éditeur de code assembleur + boutons (assembler, step, run, reset).

CustomMenuBar.java : barre de menu.

Toutes les classes du projet travaillent ensemble pour reproduire fidèlement le fonctionnement du microprocesseur 6809. Leur organisation modulaire assure une simulation claire, pédagogique et facile à utiliser.



Test de programme

Maintenant que toutes les classes ont été présentées, nous allons passer à la phase de test du programme pour vérifier son bon fonctionnement.

ETAPE 1 : Écriture du code assembleur

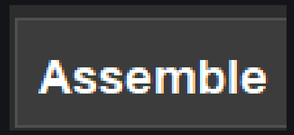
Saisir le programme dans l’éditeur :

ASSEMBLY EDITOR

```
1 ; Sample Program
2 LDA #$05
3 ADDA #$03
4 STA $20
5 END
```

ETAPE 2 : Assemblage du code

- Cliquer sur le bouton Assemble pour traduire le code assembleur en langage machine.
- Vérifier qu’aucune erreur n’est signalée.



ETAPE 3 : Chargement en mémoire

- Le programme est automatiquement placé en mémoire RAM à partir de l’adresse 0x0000.
- Les instructions apparaissent dans la table mémoire (colonnes Hex et Binaire).

ETAPE 4 : Exécution pas-à-pas (Step)

- Utiliser le bouton Step pour exécuter chaque instruction une par une.
- Observer la mise à jour des registres (A, B, PC, CC) et des drapeaux.

CPU ARCHITECTURE

LDA #$05

PC 0002

DP 00

EFHINZVC 00010000

ACCUMULATORS

A 05 B 00

D 0500

INDEX REGISTERS

X 0000 Y 0000

STACK POINTERS

S 0000 U 0000

RAM (0000-7FFF)

Address	Hex	Binary
$0000	86	10000110
$0001	05	00000101
$0002	8B	10001011
$0003	03	00000011
$0004	97	10010111
$0005	20	00100000
$0006	00	00000000
$0007	00	00000000
$0008	00	00000000
$0009	00	00000000
$000A	00	00000000
$000B	00	00000000
$000C	00	00000000

ROM (8000-FFFF)

Address	Hex	Binary
$8000	00	00000000
$8001	00	00000000
$8002	00	00000000
$8003	00	00000000
$8004	00	00000000
$8005	00	00000000
$8006	00	00000000
$8007	00	00000000
$8008	00	00000000
$8009	00	00000000
$800A	00	00000000
$800B	00	00000000
$800C	00	00000000

ASSEMBLY EDITOR

```
1 ; Sample Program
2 LDA #$05
3 ADDA #$03
4 STA $20
5 END
```

Assemble | Step Execution | Run All | Reset CPU

File Simulation Options

CPU ARCHITECTURE

ADDA #$03

PC 0004

DP 00

EFHINZVC 00010000

ACCUMULATORS

A 08 B 00

D 0800

INDEX REGISTERS

X 0000 Y 0000

STACK POINTERS

S 0000 U 0000

RAM (0000-7FFF)

Address	Hex	Binary
$0000	86	10000110
$0001	05	00000101
$0002	8B	10001011
$0003	03	00000011
$0004	97	10010111
$0005	20	00100000
$0006	00	00000000
$0007	00	00000000
$0008	00	00000000
$0009	00	00000000
$000A	00	00000000
$000B	00	00000000
$000C	00	00000000

ROM (8000-FFFF)

Address	Hex	Binary
$8000	00	00000000
$8001	00	00000000
$8002	00	00000000
$8003	00	00000000
$8004	00	00000000
$8005	00	00000000
$8006	00	00000000
$8007	00	00000000
$8008	00	00000000
$8009	00	00000000
$800A	00	00000000
$800B	00	00000000
$800C	00	00000000

ASSEMBLY EDITOR

```
1 ; Sample Program
2 LDA #$05
3 ADDA #$03
4 STA $20
5 END
```

Assemble | Step Execution | Run All | Reset CPU

File Simulation Options

CPU ARCHITECTURE

STA $20

PC 0006

DP 00

EFHINZVC 00010000

ACCUMULATORS

A 08 B 00

D 0800

INDEX REGISTERS

X 0000 Y 0000

STACK POINTERS

S 0000 U 0000

RAM (0000-7FFF)

Address	Hex	Binary
$0000	86	10000110
$0001	05	00000101
$0002	8B	10001011
$0003	03	00000011
$0004	97	10010111
$0005	20	00100000
$0006	00	00000000
$0007	00	00000000
$0008	00	00000000
$0009	00	00000000
$000A	00	00000000
$000B	00	00000000
$000C	00	00000000

ROM (8000-FFFF)

Address	Hex	Binary
$8000	00	00000000
$8001	00	00000000
$8002	00	00000000
$8003	00	00000000
$8004	00	00000000
$8005	00	00000000
$8006	00	00000000
$8007	00	00000000
$8008	00	00000000
$8009	00	00000000
$800A	00	00000000
$800B	00	00000000
$800C	00	00000000

ASSEMBLY EDITOR

```
1 ; Sample Program
2 LDA #$05
3 ADDA #$03
4 STA $20
5 END
```

Assemble | Step Execution | Run All | Reset CPU

File Simulation Options

CPU ARCHITECTURE

END - PROGRAM HALTED

PC 0007

DP 00

EFHINZVC 00010000

ACCUMULATORS

A 08 B 00

D 0800

INDEX REGISTERS

X 0000 Y 0000

STACK POINTERS

S 0000 U 0000

RAM (0000-7FFF)

Address	Hex	Binary
$0000	86	10000110
$0001	05	00000101
$0002	8B	10001011
$0003	03	00000011
$0004	97	10010111
$0005	20	00100000
$0006	00	00000000
$0007	00	00000000
$0008	00	00000000
$0009	00	00000000
$000A	00	00000000
$000B	00	00000000
$000C	00	00000000

ROM (8000-FFFF)

Address	Hex	Binary
$8000	00	00000000
$8001	00	00000000
$8002	00	00000000
$8003	00	00000000
$8004	00	00000000
$8005	00	00000000
$8006	00	00000000
$8007	00	00000000
$8008	00	00000000
$8009	00	00000000
$800A	00	00000000
$800B	00	00000000
$800C	00	00000000

ASSEMBLY EDITOR

```
1 ; Sample Program
2 LDA #$05
3 ADDA #$03
4 STA $20
5 END
```

Assemble | Step Execution | Run All | Reset CPU

thank you