Royaume de Maroc
Univeristé Hassan premier Settat
Faculté des sciences et Techniques de Settat

Guide d'utilisation de

Moto 6809 S

Réalisé par :

Chaimaa ALIOUAN

Douae AMZIL

Hassan AMRANI

Hasnae EL JABARI

Encadré par :

Pr. Hicham BENALLA

Présentation du Logiciel MOTO6809 :

Le logiciel **MOTO6809** est un environnement de développement conçu pour simuler et déboguer les programmes écrits en langage d'assemblage pour le processeur **Motorola6809**. Principalement destiné aux étudiants en électroniques et micro-informatiques, ainsi qu'aux enseignants et laboratoires électroniques, il offre plusieurs fonctionnalités.

Ce logiciel permet de configurer la taille de la mémoire ROM et RAM, de définir les adresses internes du processeur. Il propose différents modes de simulation tels que l'exécution pas-à-pas, une fonctionnalité de simulation arrière et des points d'arrêt pour une analyse détaillée de code.

Il simule également divers types d'interruptions, offre des options d'impression paramétrables, intègre un éditeur pour écrire et éditer du code assembleur, permet l'affectation de valeurs aux cases mémoires.

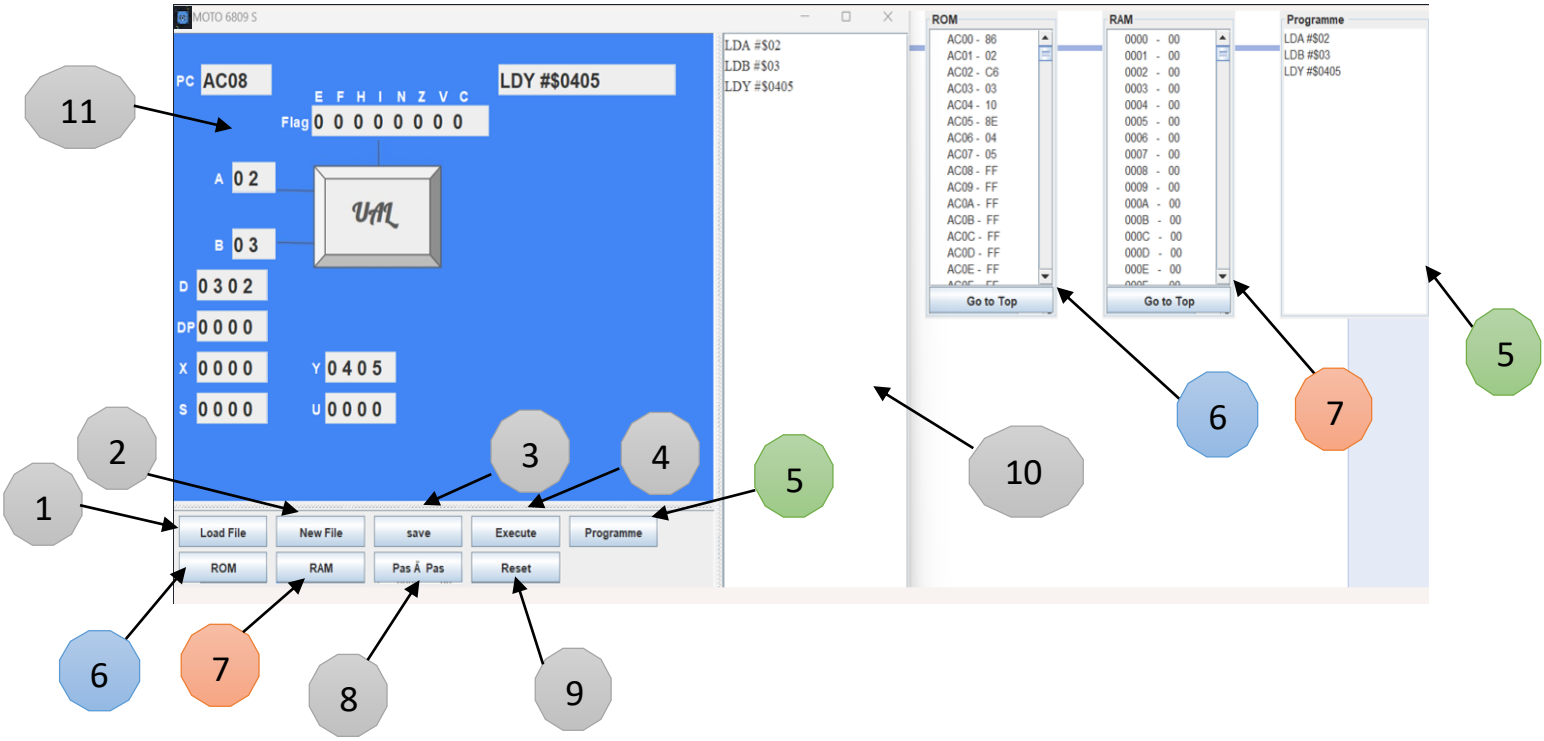
Le simulateur de microprocesseur Motorola 6809 :

Le projet **Moto 6809 S** vise à développer un simulateur pour le processeur Motorola 6809. Il s'agit de créer une plateforme d'émulation solide, offrant un environnement propice au développement, aux tests et à l'apprentissage des programmes spécifiquement conçus pour ce microprocesseur.

Moto 6809 S vise à être un outil précieux pour les programmeurs, les étudiants et les passionnés d'électronique, leur permettant d'explorer et de perfectionner leurs compétences dans le langage d'assemblage du 6809 et de réaliser des tests approfondies sans recourir à du matériel physique dédié.

L'environnement de développement de **Moto 6809 S** est riche en fenêtres et éléments de tous types.

Lorsque toutes les fenêtres sont ouvertes, le programme se présente de la façon suivante :



Pour la création de cette interface, on a commencé par la création de la fenêtre principale de titre **moto 6809 S** qui comporte 2 panneaux : d'une part, Un panneau à droite dédié à l'éditeur de texte où l'utilisateur pourra rédiger le programme. D'autre part, un panneau situé à gauche affiche l'architecture interne du 6809 avec les différents registres lors de la simulation. Ce dernier panneau est accompagné d'une barre de boutons pour faciliter l'accès aux fonctionnalités principales.

○ 1. Load File :

Le bouton « **Load File** » (charger Fichier) est utilisé pour permettre à l'utilisateur de charger un fichier contenant des programmes préalablement enregistrés.

Lorsque ce bouton est activé, une boite de dialogue ou une interface s'ouvre, permettant à l'utilisateur de parcourir les fichiers du système, et de sélectionner le fichier qu'il souhaite charger.

Une fois le fichier choisi, son contenu est chargé dans l'éditeur de texte, cela permet à l'utilisateur d'exécuter le programme, de le modifier directement dans l'éditeur de texte.

2. New File :

Le bouton « **New File** » (Nouveau Fichier) est utilisé pour créer un nouveau espace de travail dédié à l'écriture de code assembleur.

En appuyant sur ce bouton, cela pourrait effacer le contenu actuel de l'éditeur de texte, offrant à l'utilisateur un espace vierge pour commencer à écrire un nouveau code assembleur.

Cela permettrait à l'utilisateur de démarrer un nouveau programme à partir de zéro, en ayant un espace de travail propre et prêt à recevoir du nouveau code.

3. Save :

Le bouton « **Save** »(Enregistrer) est utilisé pour sauvegarder les modifications apportées à un fichier.

En cliquant sur ce bouton, les changements effectués dans le fichier en cours d'édition (programme assembleur) seront sauvegardées si le fichier a été enregistré précédemment. Sinon, une boite de dialogue s'ouvre généralement, permettant à l'utilisateur de lui donner un nom, après avoir attribué un nom au fichier, en cliquant sur « Save », les modifications apportées dans le programme assembleur existant dans l'éditeur de texte seront enregistrées.

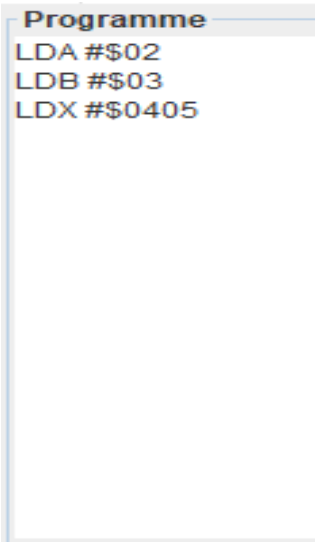
4. Execute :

Le bouton « **Execute** » permet d'exécuter le programme assembleur à partir de la position actuelle du compteur de programme jusqu'à ce qu'il atteigne la fin du programme ou jusqu'à un point d'arrêt s'il est défini.

Il est important de souligner qu'avant d'exécuter un programme assembleur présent dans l'éditeur de texte, il est nécessaire de l'enregistrer en cliquant sur le bouton « **save** ». Cette étape garantit que les modifications apportées au code sont sauvegardées dans un fichier avant de procéder à son exécution.

5. Le bouton & La fenêtre PROGRAMME :

Comme son nom l'indique, cette fenêtre affiche le programme assembleur courant.

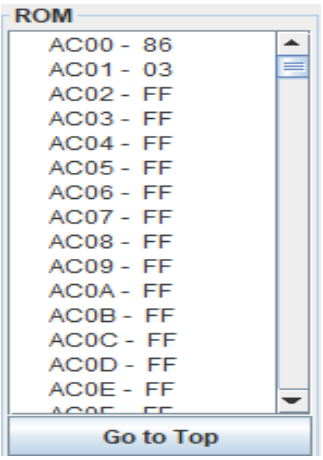


Cette fenêtre facilite le transfert du programme saisi par l'utilisateur depuis l'éditeur de texte vers la fenêtre Programme. Chaque fois que des modifications sont apportées dans l'éditeur de texte, un simple clic sur le bouton **« Programme** » permet d'afficher instantanément le programme actualisé dans la fenêtre Programme.

Il est important de noter que cette fenêtre n'est pas modifiable par l'utilisateur. Pour apporter des modifications. L'utilisateur doit modifier le contenu dans l'éditeur de texte.

6. Le bouton & La fenêtre ROM :

La fenêtre ROM affiche l'espace de mémoire morte ainsi que le contenu des différentes cases de cette mémoire.



Chaque ligne comporte :

- à gauche l'adresse sur 16bits.
- au milieu, une barre verticale de séparation.
- à droite, la donnée placée dans la case mémoire.

Cet espace mémoire comporte les codes hexadécimaux correspondants aux instructions du programme.

C'est une mémoire en lecture seule, ce qui signifie que les données stockées à l'intérieur ne peuvent être modifiées qu'après l'exécution d'un programme.

Les valeurs contenues dans cette ROM demeurent constantes pendant l'exécution du programme et ne peuvent être altérées qu'à travers des processus de reprogrammation ou de mise à jour qui impliquent des actions spécifiques hors de l'exécution normale du programme.

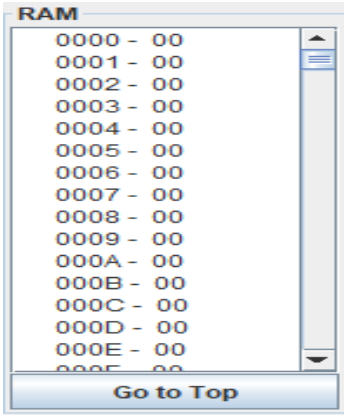
Le bouton « **Go to Top** » facilite la navigation pour l'utilisateur en lui permettant de remonter rapidement au début ou au sommet de cette fenêtre.

La fenêtre ROM peut être cachée ou affichée en cliquant sur le bouton « **ROM** ».

Lorsque la fenêtre **ROM** est affichée et lorsque l'utilisateur clique sur le bouton **ROM**, la fenêtre ROM se masque d'abord. Ensuite, la fonction « **Go to Top** » est activée pour ramener l'utilisateur au début de la liste de la ROM, après la fenêtre ROM se réaffiche.

○ 7. Le bouton & La fenêtre RAM :

Comme son nom l'indique, la fenêtre « **RAM** » affiche l'espace de mémoire vive ainsi que le contenu des différentes cases de cette mémoire.



Chaque ligne comporte ;

- à gauche, Une adresse sur 16 bits.
- au milieu, Une barre verticale de séparation.
- à droite, la donnée placée dans la case mémoire.

Le bouton « **Go to Top** » facilite la navigation pour l'utilisateur en lui permettant de remonter rapidement au début de cette fenêtre.

En cliquant sur le bouton « **RAM** », l'utilisateur peut afficher ou masquer la fenêtre **RAM**.

Si la fenêtre **RAM** est déjà affichée et que l'utilisateur clique à nouveau sur le bouton, la fenêtre se cache. Ensuite, la fonction « **Go to Top** » est automatiquement activée pour ramener l'utilisateur au début de la liste de la **RAM**, après cette action, la fenêtre RAM se réaffiche.

○ 8. Pas-à-pas :

Le bouton « **Pas-à-pas** » permet d'avancer le programme assembleur d'une seule instruction à la fois.

En utilisant cette fonctionnalité, chaque fois que vous appuyez sur le bouton « **Pas-à-pas** », le programme exécute une seule instruction, puis s'arrête, vous permettant d'examiner les résultats de cette instruction spécifique. Cela permet une exécution contrôlée et étape par étape du programme, offrant à l'utilisateur la possibilité d'observer le fonctionnement du programme assembleur ligne par ligne ou instruction par instruction.

○ 9. Reset :

Le bouton « **Reset** » permet de réinitialiser le programme en remettant le compteur de programme à l'adresse de la première instruction du programme.

En utilisant cette fonctionnalité, lorsque vous appuyez sur le bouton « **Reset** », le programme revient à son état initial. Le compteur de programme, qui indique où se trouve l'exécution du programme, est ramené à l'adresse de la première instruction du programme. Cela équivalent à redémarrer ou réinitialiser le programme depuis le début.

○ 10. Editeur de texte :

L'éditeur de texte fournit une interface où les utilisateurs peuvent écrire, éditer et manipuler du programme assembleur.

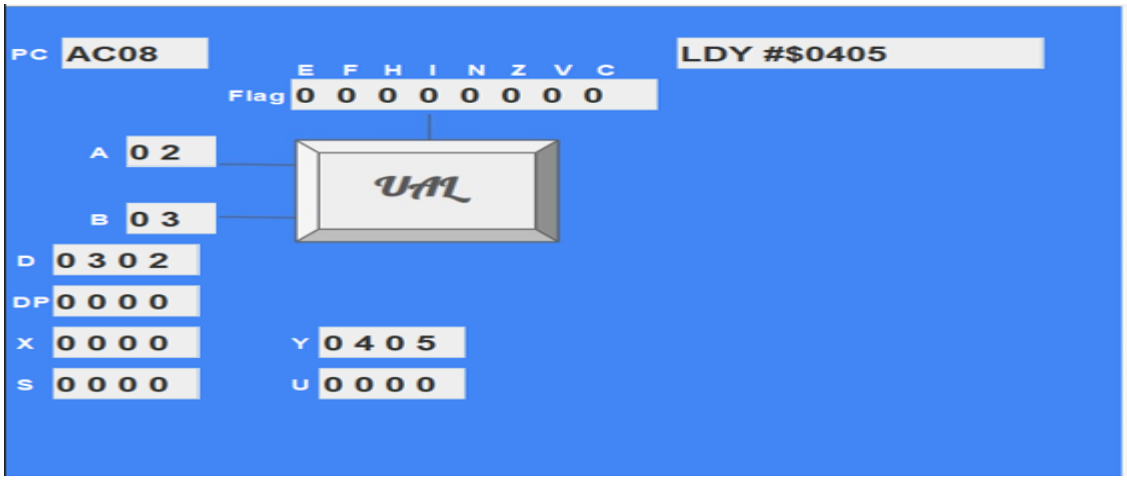
Il offre une fenêtre de travail généralement vide au départ, permettant aux utilisateurs de saisir du contenu.

Cette interface est associée à la fenêtre principale, reste toujours visible, offrant ainsi un accès rapide et constant à l'outil d'édition du code source.

○ 11. La fenêtre Architecture interne du 6809 :

Le 6809 est un microprocesseur 8 bits à architecture interne 16bits.

Cette fenêtre comprend plusieurs registres essentiels qui jouent des rôles spécifiques dans l'exécution des programmes.



Accumulateur A et B : L'accumulateur A est principalement utilisé pour effectuer des opérations arithmétiques et logiques. Le registre B est une extension de l'accumulateur A pour des opérations nécessitant des données sur 16 bits.

Registre de programme (PC-Program Counter) : il contient l'adresse de la prochaine instruction à exécuter dans le programme.

Pointeurs de pile S et U : leur taille est de 16 bits. Ces registres permettent d'avoir deux piles distinctes. Habituellement, l'une est utilisée par défaut pour les instructions de sous-programmes tandis que l'autre est à la disposition de l'utilisateur pour divers taches telles que le passage de paramètres.

Registre de Directe (DP – Direct Register) il est de 8 bits et permet de diviser l'espace mémoire en 256 pages de 256 octets. Ce registre est utilisé pour l'adressage direct, facilitant l'accès à différentes zones de la mémoire.

Registres d'index X, Y : ils agissent comme des pointeurs d'adresses et possèdent une taille de 16bits, similaire à celle du bus d'adresse.

Cette fenêtre affiche le contenu de ces registres et permet également à l'utilisateur de modifier temporairement les valeurs de certains de ces registres.

Toutefois, il est important de noter que ces modifications ne sont pas permanentes.

Les valeurs des registres sont généralement modifiées via le programme assembleur saisi dans l'éditeur de texte. Après avoir saisi ou modifié le programme dans l'éditeur de texte, vous pouvez l'enregistrer et l'exécuter pour que les valeurs des registres soient mises-à-jour en conséquence.

Cette fenêtre est associée à la fenêtre principale, reste toujours visible et permet à l'utilisateur de visualiser et de comprendre l'état des registres pendant la simulation, ce qui peut être utile pour analyser le comportement du programme en cours d'exécution.

Jeu d'instruction :

Format du Programme :

- Chaque ligne doit contenir une seule instruction (mnémonique + opérande), ou une étiquette suivie de « : », ou un commentaire.

- Un commentaire peut suivre une instruction à conditions qu'il est précédé du caractère « ; » ou « * » et qu'il ne comporte pas le caractère « : ».

- Les adresses-opérandes doivent avoir exactement 4 caractères précédés du symbole « $ ».

- Les données-opérandes doivent comporter exactement 2 caractères précédés du symbole « $ ».

- La pseudo-instruction **DB** sera placée de préférence en début du programme. Cette instruction est souvent utilisée pour définir des données.

- Le programme doit se terminer par la pseudo-instruction **END** pour indiquer explicitement la fin du programme.

○ Modes d'adressages :

Les **modes d'adressages** définissent comment les instructions accèdent et manipulent les données ou les emplacements mémoire lors de l'exécution du programme.

Les modes d'adressages utilisés dans Ce logiciel sont :

✓ Mode d'adressage inhérent :

- Ce mode d'adressage concerne les instructions qui n'ont pas besoin d'opérande explicite car elles utilisent implicitement des registres ou des emplacements mémoire prédéfinis dans le processeur.

- Ces instructions agissent directement sur des données ou des registres internes au processeur sans nécessiter de spécification d'opérande. Par exemple, certaines opérations arithmétiques comme **CLR** (Clear).

✓ Mode d'adressage Etendu > :

Ce mode d'adressage est utilisé pour accéder à des adresses mémoire plus grandes. Permettant d'atteindre des emplacements mémoire au-delà de 256 octets. Par exemple, **LDA >$3000**, indiquerait l'accès à une adresse mémoire au-delà de la limite des 256 premiers octets.

✓ Mode d'adressage Immédiat # :

- Dans ce mode, l'opérande spécifié est une valeur ou une donnée directement utilisée par l'instruction.

- L'opérande immédiat est fourni directement dans le code de l'instruction. Il peut s'agir d'une valeur décimale ou hexadécimale. Par exemple, dans l'instruction **LDA #$32**, #$32 est l'opérande immédiat qui charge la valeur hexadécimale 32 dans l'accumulateur A.

○ Liste des instructions :

LDA (Load Accumulateur A) : Charge une valeur depuis un emplacement mémoire spécifié dans l'accumulateur A.
Exemple : LDA #$07 ; #$07 est la valeur immédiate à charger dans l'accumulateur A.

LDB (Load Accumulateur B) : Charge une valeur depuis un emplacement mémoire spécifié dans l'accumulateur B.
Exemple : LDB #$42 ; #$42 est la valeur immédiate à charger dans l'accumulateur B.

LDD (Load Double Accumulateur) : Chargement de D avec le contenu mémoire (D contient deux registres 8 bits).
Exemple : LDD #$1234 ; #$1234 est la valeur à charger dans l'accumulateur D.
La même procédure pour **LDX, LDY, LDU, LDS.**

STA (Store Accumulator A) : Stocke le contenu de l'accumulateur dans l'emplacement mémoire spécifié par l'opérande.
Exemple : STA >$3000 ; pour stocker la valeur de l'accumulateur A à l'adresse mémoire $3000.

STB : même procédure que **STA,** c'est de stocker le contenu du B à une adresse mémoire spécifiée.

STD : Stocke le registre de données (D) dans la mémoire à l'adresse spécifiée.
Exemple : STD >$2000 ; stocke la valeur du registre D à l'adresse mémoire $2000
Le même procédure pour les autres registres **U (STU), X (STX), Y (STY).**

SUBA (Substract from Accumulateur A) : Soustrait la valeur d'un registre ou d'une mémoire d'accumulateur A.
La même procédure pour les autres registres **B (SUBB), D (SUBD).**

ADCA (Add accumulator A with Carry) : Ajoute la valeur de l'accumulateur A avec retenue à une donnée spécifiée.
Exemple : ADCA #$10 ; ajoute la valeur $10 à l'accumulateur A avec retenue.

ADCB : même procédure que **ADCA,** pour l'accumulateur B.

ADDA (Add to Accumulator A) : ajouter une donnée spécifiée à l'accumulateur A.

Exemple : ADDA #$40 ; ajoute la valeur $40 à l'accumulateur A.

ADDB : même procédure que **ADDA**, pour l'accumulateur B.

ADDD : addition du contenu mémoire au registre D (16bits).
Exemple : ADDD #$1234 ; ajoute la valeur $1234 au registre D.

ANDA (Logical AND with Accumulator A) : Effectue un ET logique entre l'accumulateur A et une donnée spécifiée.
Exemple : ANDA #$0F ; réalise un ET logique entre l'accumulateur A et la valeur $0F.

ANDB : même procédure que **ANDA**, pour l'accumulateur B.

BITA (Tests Bits of Accumulator A) : teste les bits de l'accumulateur A avec une donnée spécifiée.
BITB : même procédure que **BITA**, pour l'accumulateur B.

EORA (Exclusive OR with Accumulateur A) : Effectue un OU exclusif entre l'accumulateur A et une donnée spécifiée.
Exemple : EORA #$55 ; réalise un OU exclusif entre l'accumulateur A et la valeur $55.
EORB : même procédure sur **EORA**, pour l'accumulateur B.

ORA (Logical OR with Accumulateur A) : Effectue un OU logique entre l'accumulateur A et une donnée spécifiée.
Exemple : ORA #$F0 ; réalise un OU logique entre l'accumulateur A et la valeur $0F.
ORB : même procédure que **ORA**, pour l'accumulateur B.

CLRA (Clear Accumulator A) : Efface (met à zéro) le contenu de l'accumulateur A.

CLRB (Clear Accumulator B) : Efface (met à zéro) le contenu de l'accumulateur B.

COMA (Complement Accumulator A) : Complémente (inverse) chaque bit de l'accumulateur A.

COMB (Complement Accumulator B) : Complémente (inverse) chaque bit de l'accumulateur B.

INCA (Increment Accumulator A) : Incrémente (ajoute 1) à la valeur de l'accumulateur A.

DECA (Decrement Accumulator A): Décrémente (Soustrait 1) de la valeur de l'accumulateur A.

INCB (Increment Accumulator B) : Incrémente (ajoute 1) à la valeur de l'accumulateur B.

DECB (Decrement Accumulator B) : Décrémente (Soustrait 1) de la valeur de l'accumulateur B.

CMPA : Compare le contenu du registre A (Accumulateur A) avec le contenu mémoire.

Même procédure pour les autres registres : **B (CMPB), D (CMPD), S (CMPS), U (CMPU), X (CMPX), Y (CMPY).**

PSHS (Push on Stack) : Empilement de registres sur S.
Exemple : PSHS A, B, X ; Empile les contenus des registres A, B, X, sur le pile système, pointée par le registre S.

ROLA (Rotate Left A) : Effectue un décalage circulaire (une rotation) à gauche sur le contenu d'Accumulateur A.

ROLB (Rotate Left B) : Effectue un décalage circulaire (une rotation) à gauche sur le contenu d'Accumulateur B.

RORA (Rotate Right A) : Effectue une rotation vers la droite sur le contenu d'Accumulateur A.

RORB (Rotate Right B) : Effectue une rotation vers la droite sur le contenu d'Accumulateur B.

LSLA (Logical Shift Left A) : Effectue un décalage logique vers la gauche sur le contenu d'Accumulateur A.

LSLB (Logical Shift Left B) : Effectue un décalage logique vers la gauche sur le contenu d'Accumulateur B.

LSRA (Logical Shift Right A) : Effectue un décalage logique vers la droite sur le contenu d'Accumulateur A.

LSRB (Logical Shift Right B) : Effectue un décalage logique vers la droite sur le contenu d'Accumulateur B.

SWI (Software Interrupt) : Déclenche une interruption logicielle.

Exemples de Simulation de Logiciel :

1er exemple :

Programme Assembleur :

```
LDA #$EF
LDB #$34
LDY #$0055
LDA #$3D
ADDA #$0F
```

Explications :

LDA #$EF : charge la valeur hexadécimale 'EF' dans l'accumulateur A.

→ Accumulateur A (A) : 'EF'

LDB #$34 : Charge la valeur hexadécimale '34' dans l'accumulateur B.

→ Accumulateur B (B) : '34'

LDY #$0055 : Charge la valeur hexadécimale '0055' dans le registre Y.

→ Registre Y (Y) : ‘0055’

LDA #$3D : Charge la valeur hexadécimale ‘3D’ dans l’accumulateur A, remplaçant la valeur précédente.

→ Accumulateur A (A) : ‘3D’

ADDA #$0F : Additionne la valeur hexadécimale ‘0F’ à l’accumulateur A.

→ Accumulateur A (A) : ‘4C ’ ; (3D + 0F = 4C (Hexadécimal)).

À la fin de ces instructions :

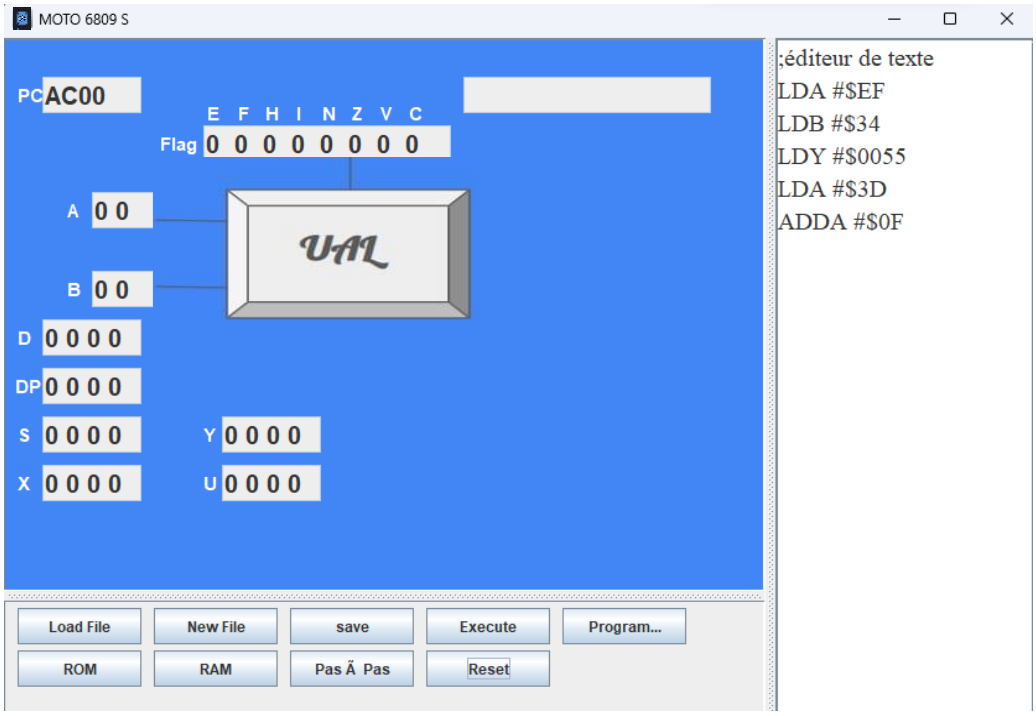
A : contient la valeur ‘4C’.

B : contient la valeur ‘34’.

Y : contient la valeur ‘0055’.

Voici la séquence d’actions qui se déroulera concrètement dans le logiciel :

1ère étape : écrire le programme dans l’éditeur de texte

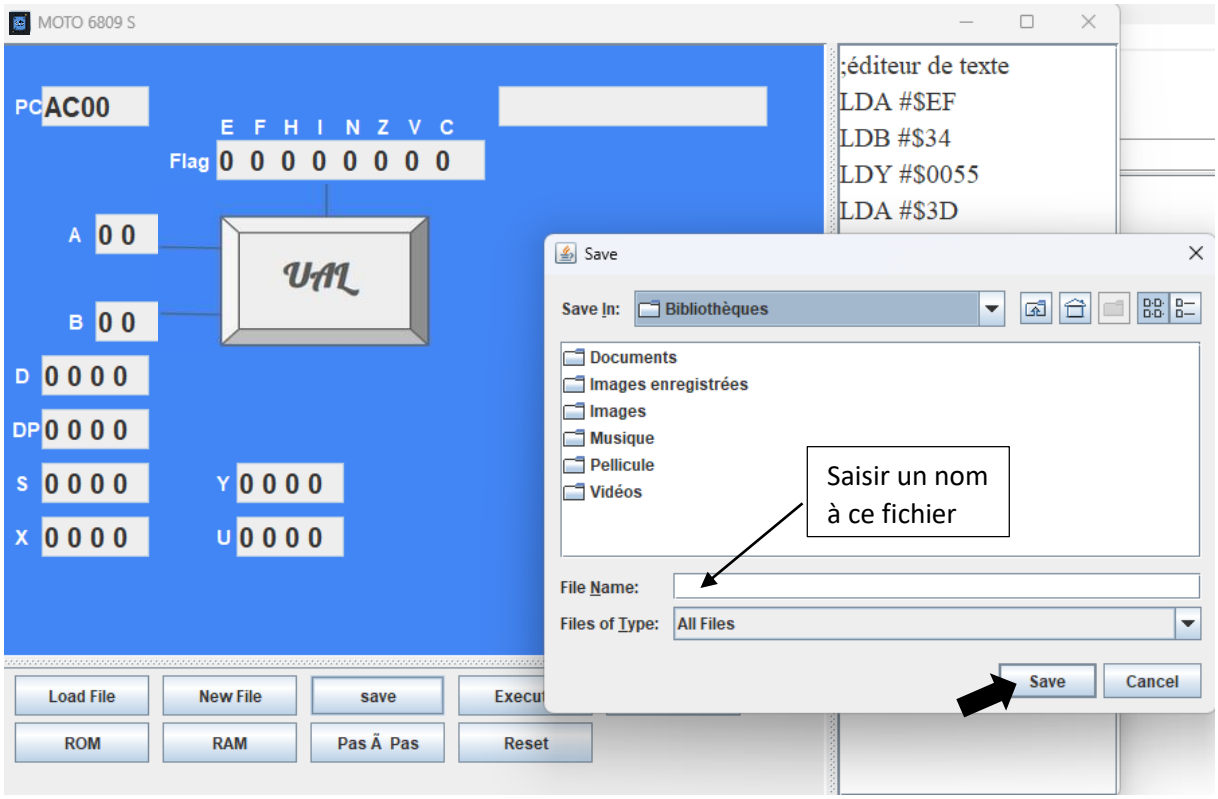


Si un programme est déjà présent dans l’éditeur de texte, vous pouvez cliquer sur le bouton « **New File** » qui donner une nouvelle interface pour saisir votre programme.

2ème étape : sauvegarder le programme :

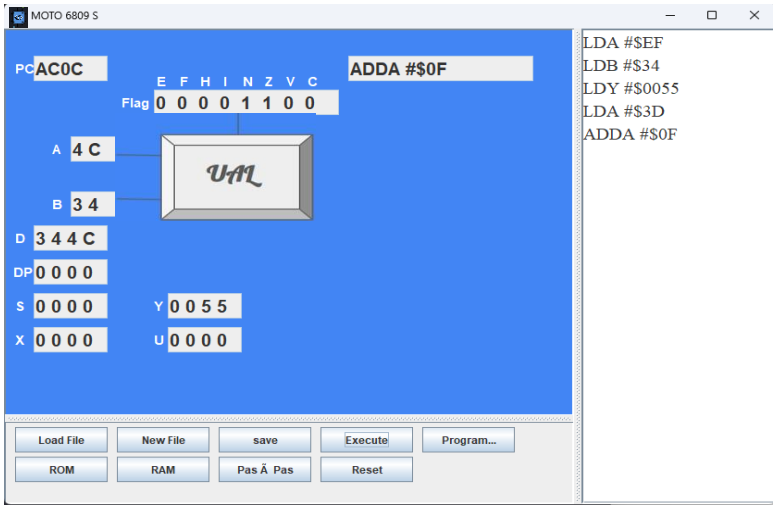
Avant d'exécuter un programme Assembleur, il faut le sauvegarder.

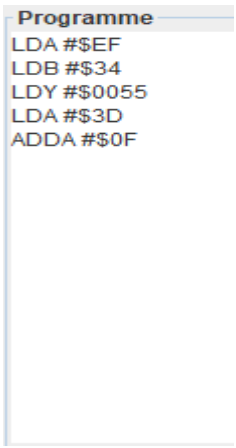
Cliquez sur le bouton « **Save** » pour enregistrer le code que vous avez saisi.



3ème étape : Exécution du programme :

Une fois que le programme a été sauvegardé, vous pouvez l'exécuter en cliquant sur le bouton « **Execute** ». Cela déclenchera l'exécution du programme que vous avez écrit dans l'éditeur de texte.



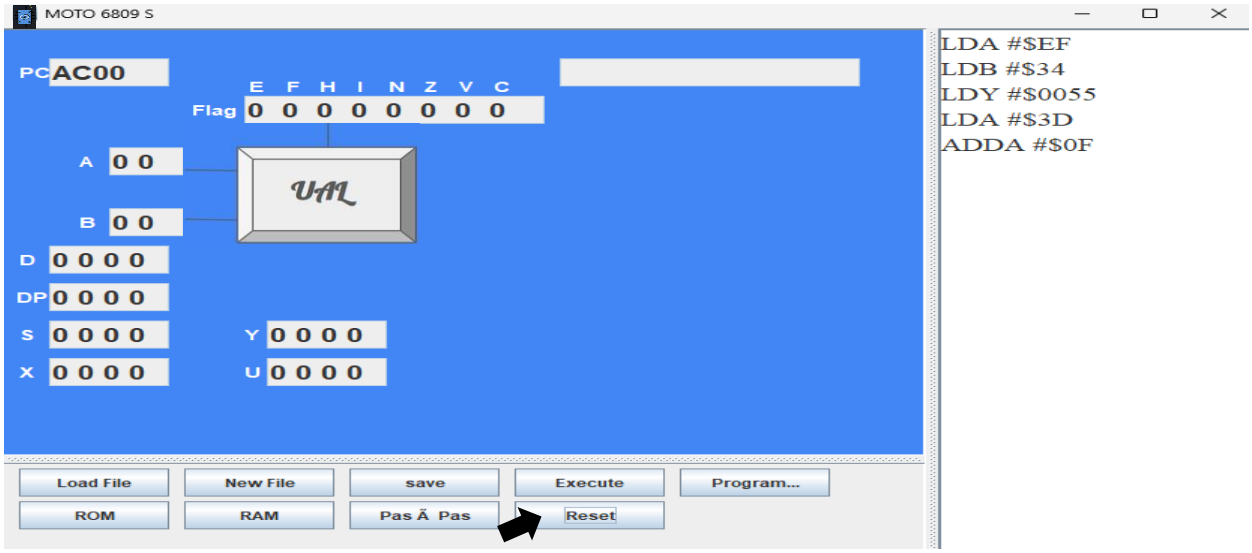


Lors de l'exécution d'un programme assembleur, chaque instruction est exécutée séquentiellement. En cas d'erreur dans une instruction, le programme peut être interrompu, affichant un message spécifiant le type d'erreur rencontré. Cela peut être dû à une faute de syntaxe, une opération invalide ou une référence mémoire incorrecte, par exemple. Cette interruption permet d'identifier et de corriger l'erreur avant de poursuivre l'exécution du programme.

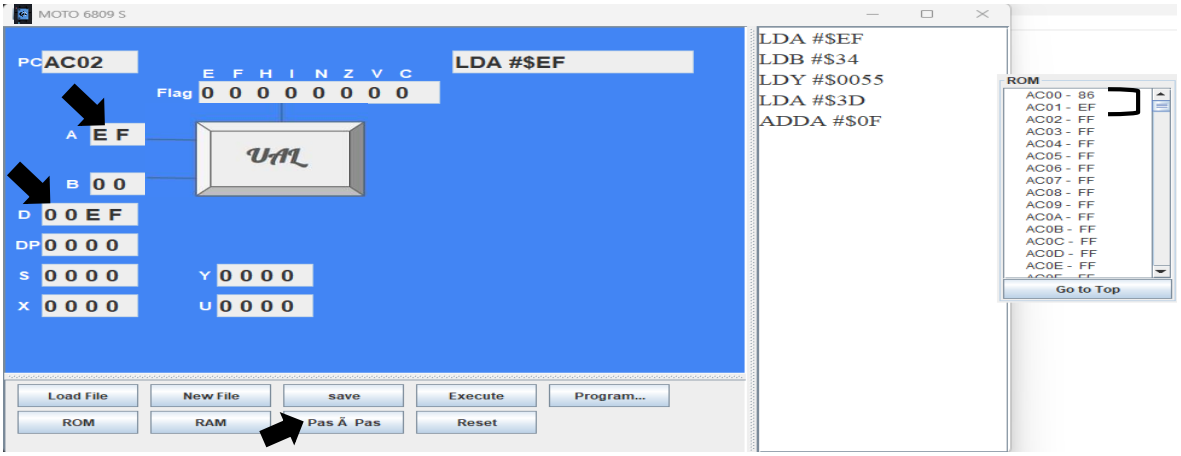
La fenêtre programme affiche également le contenu de ce programme assembleur, vous permettant ainsi de voir le code que vous avez saisi pour l'exécution.

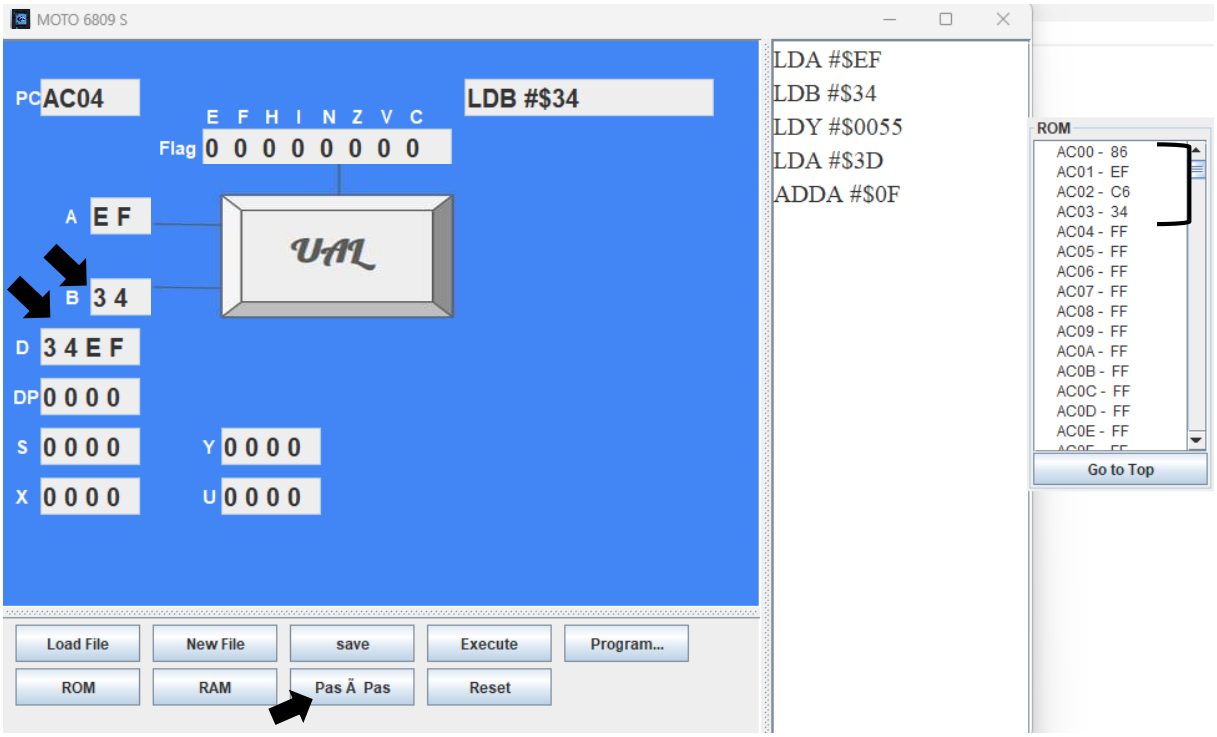
Dans la fenêtre affichant l'architecture interne du processeur 6809, vous pouvez visualiser les valeurs actuelles stockées dans les registres, ces valeurs correspondent aux mêmes valeurs trouvées théoriquement.

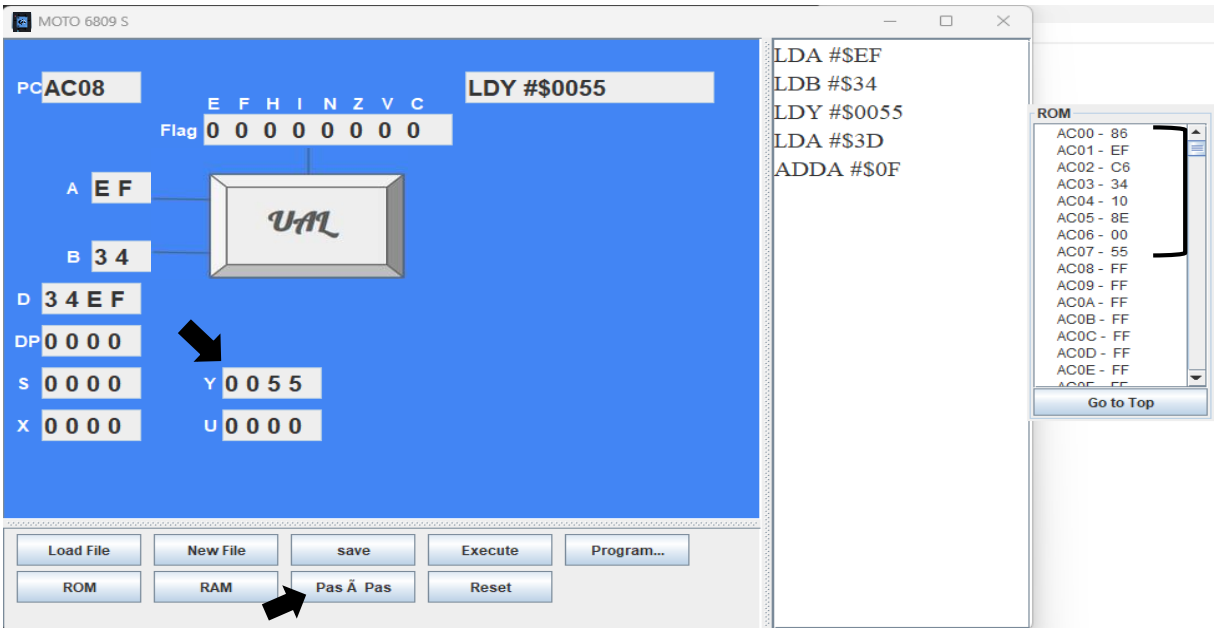
Pour exécuter le programme pas-à-pas, vous devrez d'abord remettre les registres à leur état initial en cliquant sur le bouton « Reset ».

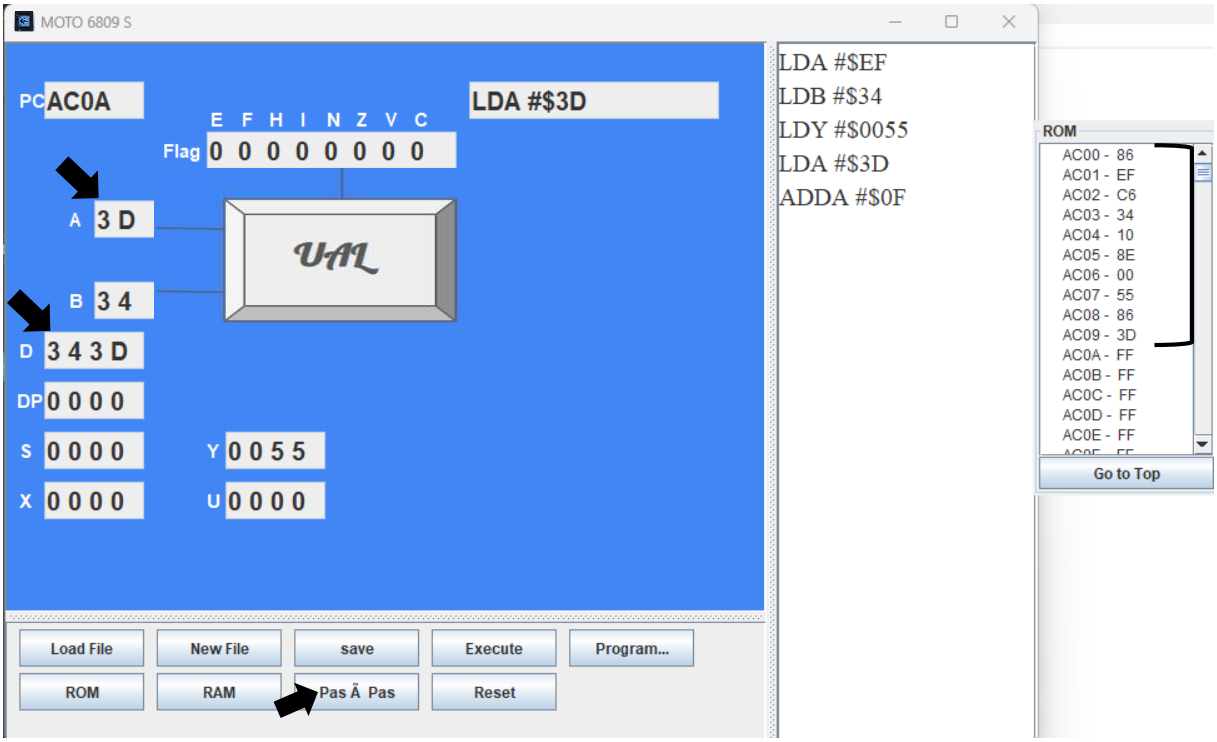


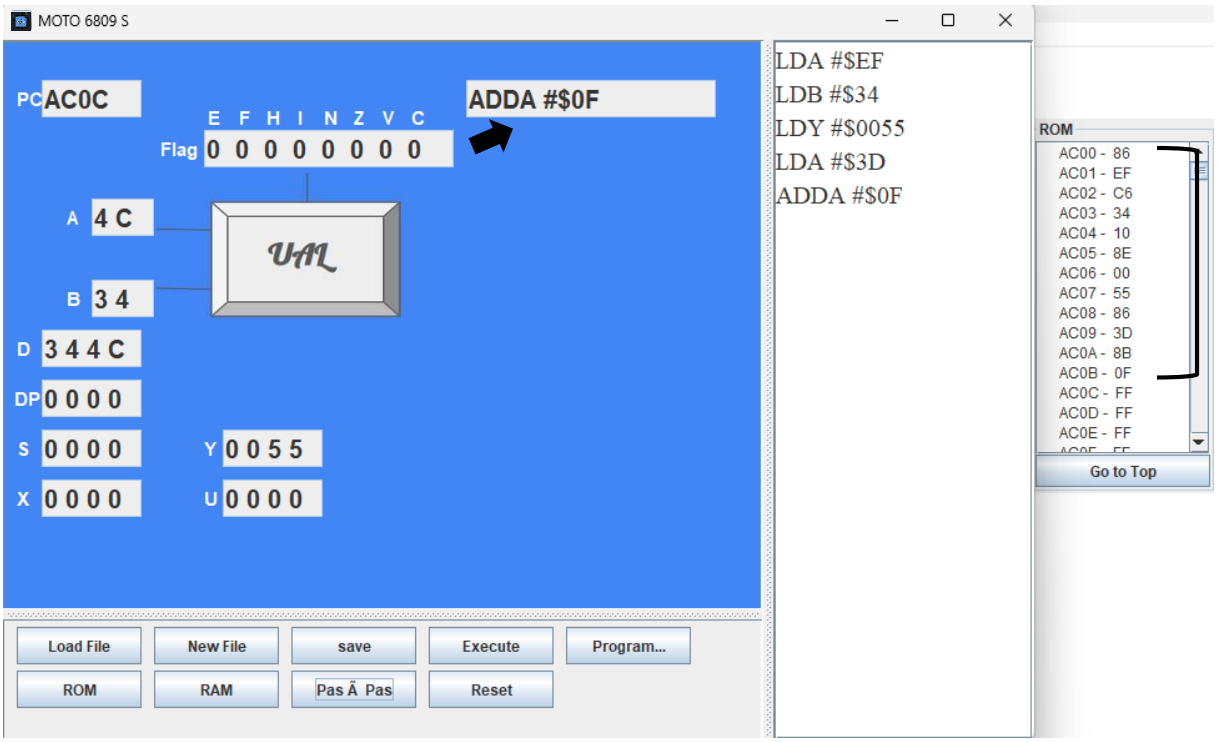
Ensuite, en cliquant sur le bouton « Pas-à-pas », vous pouvez avancer l'exécution du programme, voir les résultats à chaque étape et observer les modifications dans la mémoire ROM, ce qui vous aidera à suivre précisément le déroulement de chaque instruction du programme


MOTO 6809 S
PC AC02
LDA #$EF
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A E F
B 0 0
D 0 0 E F
DP 0 0 0 0
S 0 0 0 0
X 0 0 0 0
Y 0 0 0 0
U 0 0 0 0
UAL
LDA #$EF
LDB #$34
LDY #$0055
LDA #$3D
ADDA #$0F
ROM
AC00 - 86
AC01 - EF
AC02 - FF
AC03 - FF
AC04 - FF
AC05 - FF
AC06 - FF
AC07 - FF
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset

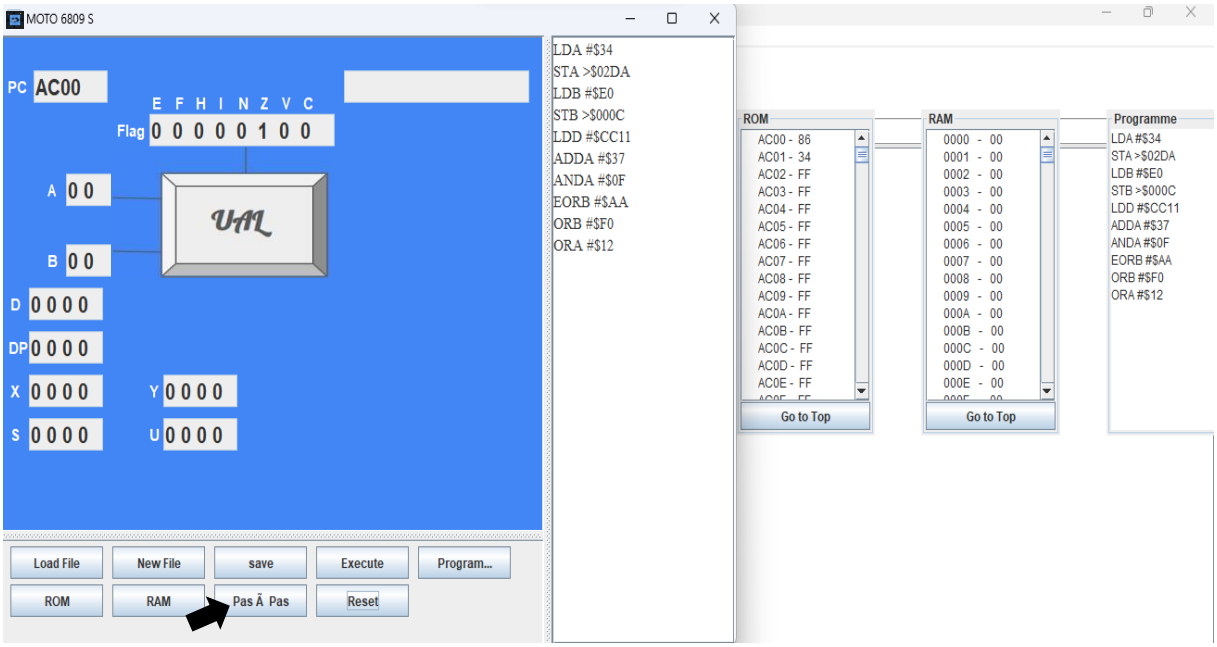

MOTO 6809 S
PC AC04
LDB #$34
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A E F
B 3 4
D 3 4 E F
DP 0 0 0 0
S 0 0 0 0
X 0 0 0 0
Y 0 0 0 0
U 0 0 0 0
UAL
LDA #$EF
LDB #$34
LDY #$0055
LDA #$3D
ADDA #$0F
ROM
AC00 - 86
AC01 - EF
AC02 - C6
AC03 - 34
AC04 - FF
AC05 - FF
AC06 - FF
AC07 - FF
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset


MOTO 6809 S
PC AC08
LDY #$0055
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A E F
B 3 4
D 3 4 E F
DP 0 0 0 0
S 0 0 0 0
X 0 0 0 0
Y 0 0 5 5
U 0 0 0 0
UAL
LDA #$EF
LDB #$34
LDY #$0055
LDA #$3D
ADDA #$0F
ROM
AC00 - 86
AC01 - EF
AC02 - C6
AC03 - 34
AC04 - 10
AC05 - 8E
AC06 - 00
AC07 - 55
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset

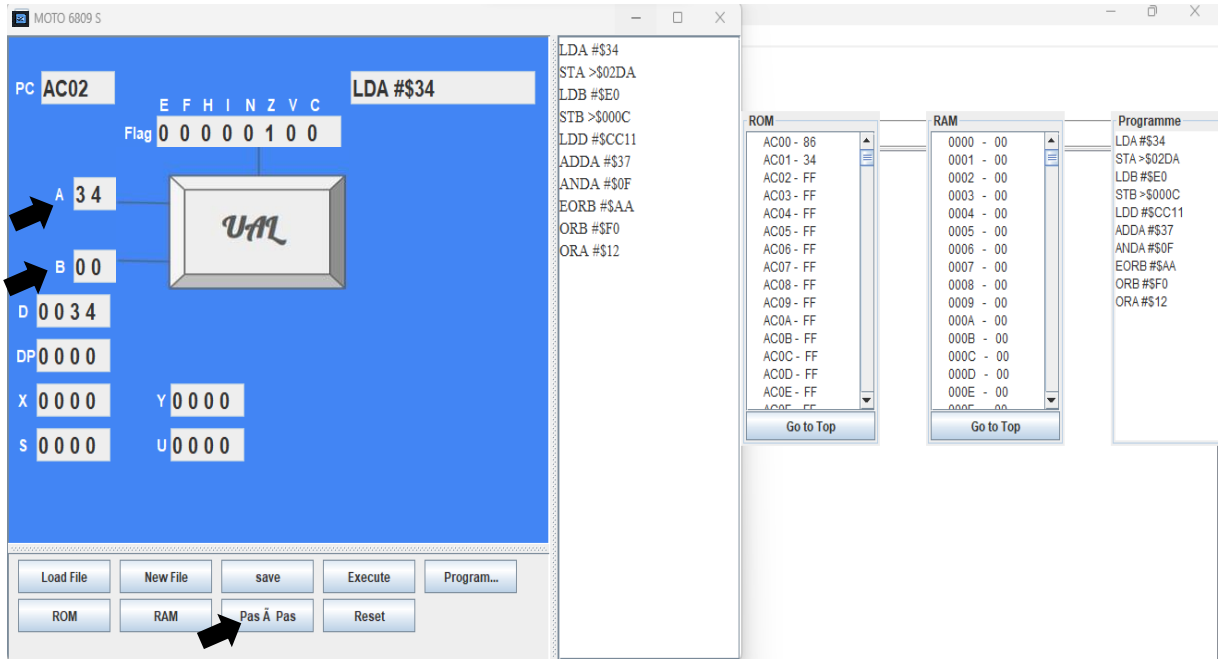

MOTO 6809 S
PC AC0A
LDA #$3D
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 3 D
B 3 4
UAL
D 3 4 3 D
DP 0 0 0 0
S 0 0 0 0
Y 0 0 5 5
X 0 0 0 0
U 0 0 0 0
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$EF
LDB #$34
LDY #$0055
LDA #$3D
ADDA #$0F
ROM
AC00 - 86
AC01 - EF
AC02 - C6
AC03 - 34
AC04 - 10
AC05 - 8E
AC06 - 00
AC07 - 55
AC08 - 86
AC09 - 3D
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top

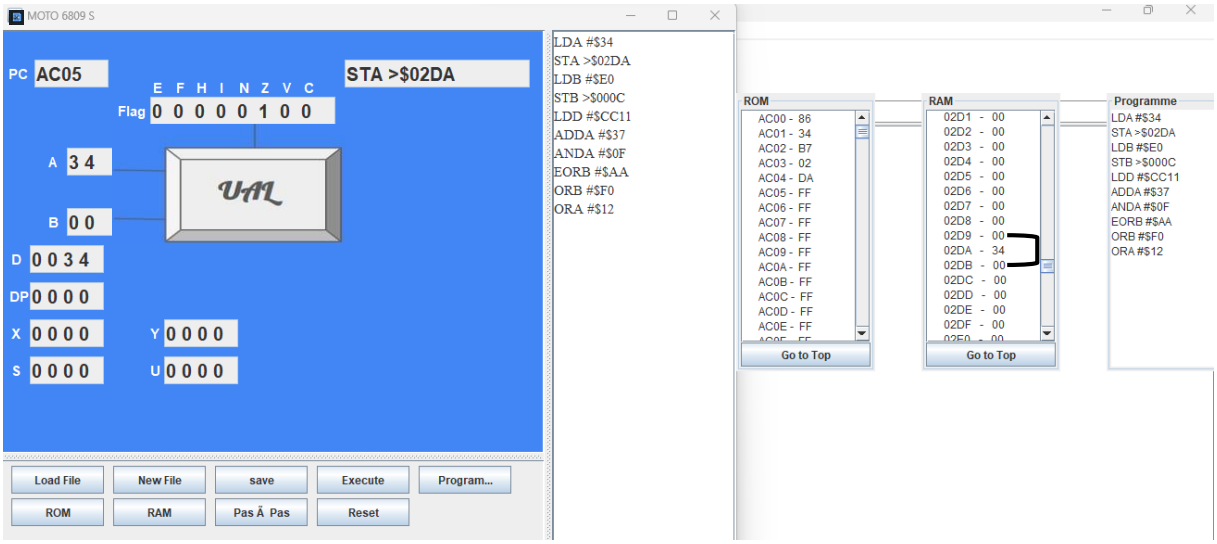

MOTO 6809 S
PC AC0C
ADDA #$0F
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 4 C
B 3 4
UAL
D 3 4 4 C
DP 0 0 0 0
S 0 0 0 0
Y 0 0 5 5
X 0 0 0 0
U 0 0 0 0
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$EF
LDB #$34
LDY #$0055
LDA #$3D
ADDA #$0F
ROM
AC00 - 86
AC01 - EF
AC02 - C6
AC03 - 34
AC04 - 10
AC05 - 8E
AC06 - 00
AC07 - 55
AC08 - 86
AC09 - 3D
AC0A - 8B
AC0B - 0F
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top

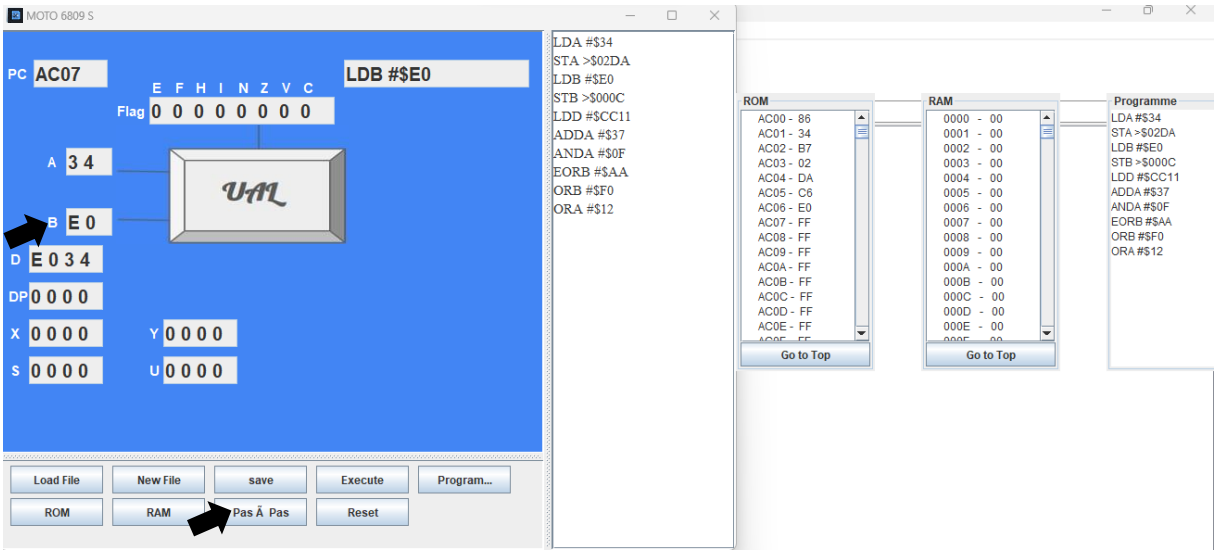
○ 2ème exemple : Le test de quelques instructions :

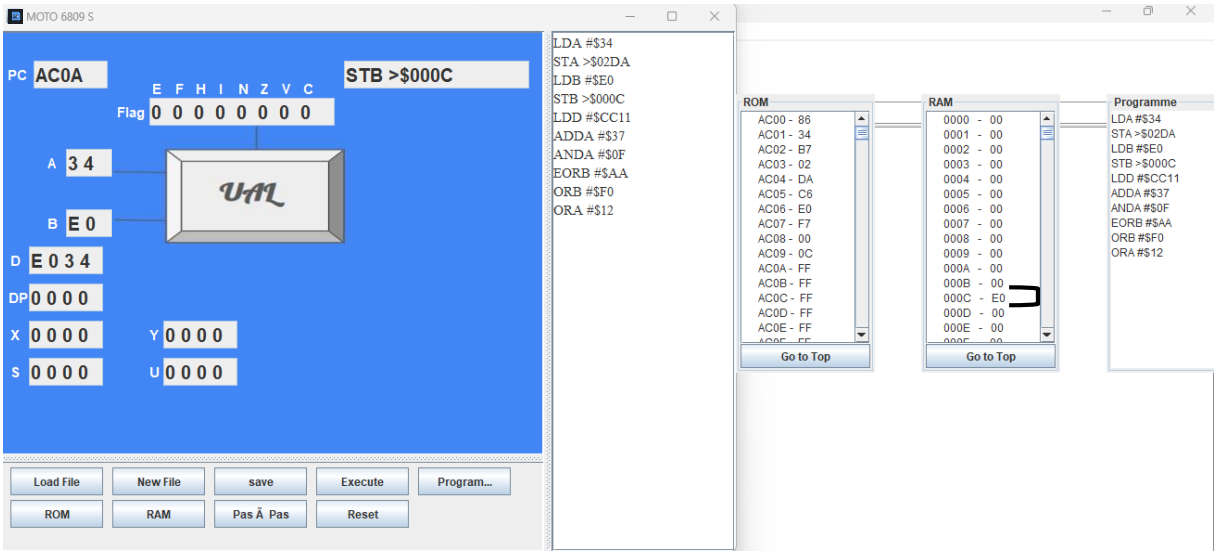


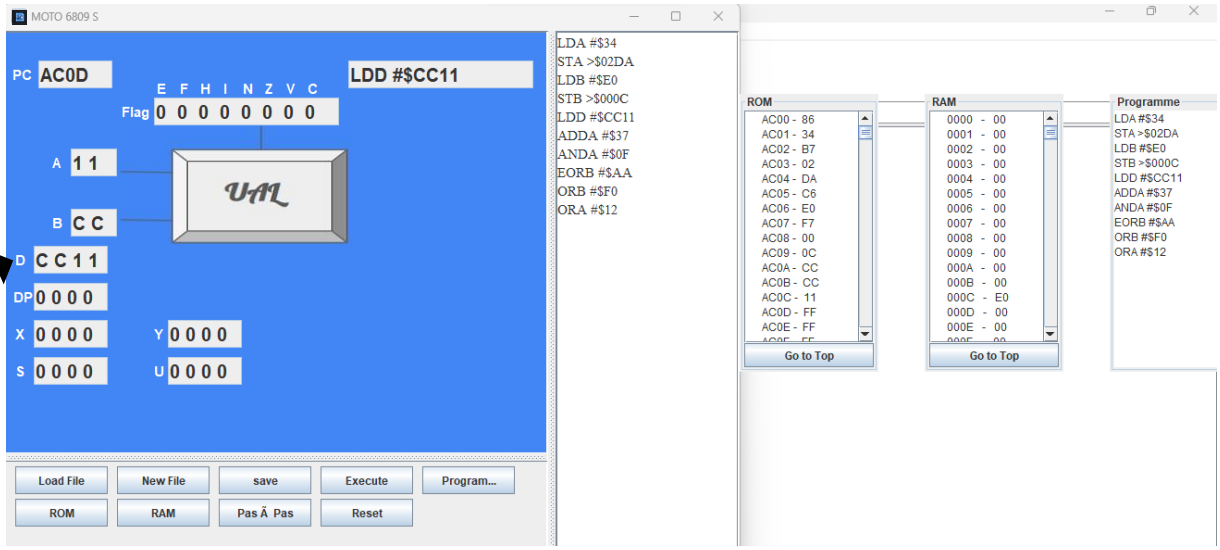
On peut cliquer soit sur le bouton « **Execute** » ou sur le bouton « Pas-à-Pas ». Nous allons cliquer sur « **Pas-à-pas** » pour poursuivre les étapes, étape par étape, instruction par instruction. Après l'exécution de chaque instruction, pour passer à l'instruction suivante, il faut cliquer une nouvelle fois sur le bouton **« Pas-à-pas** ».

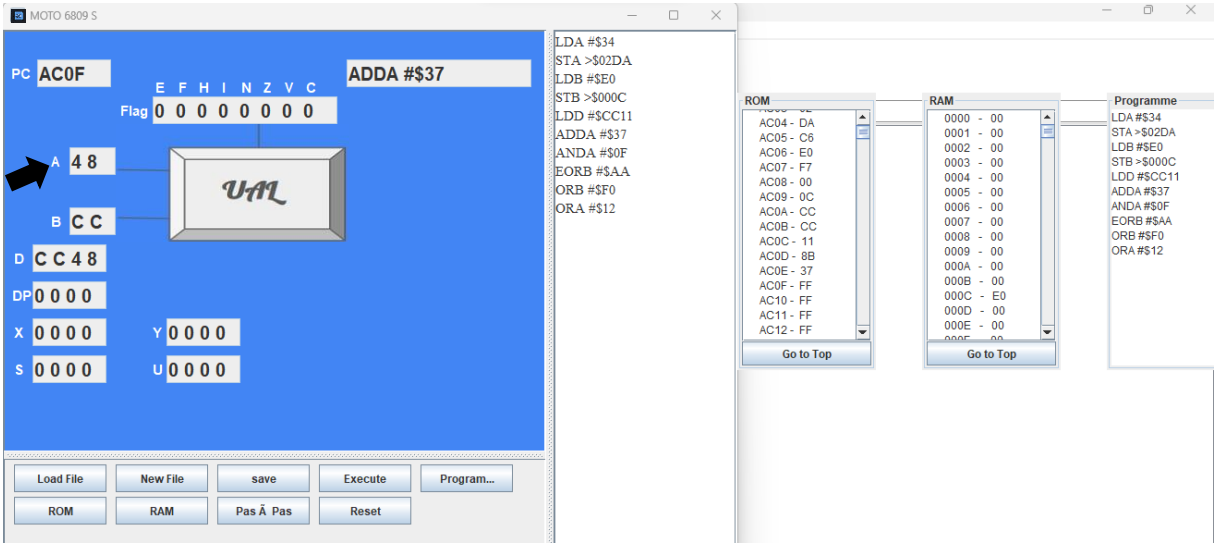


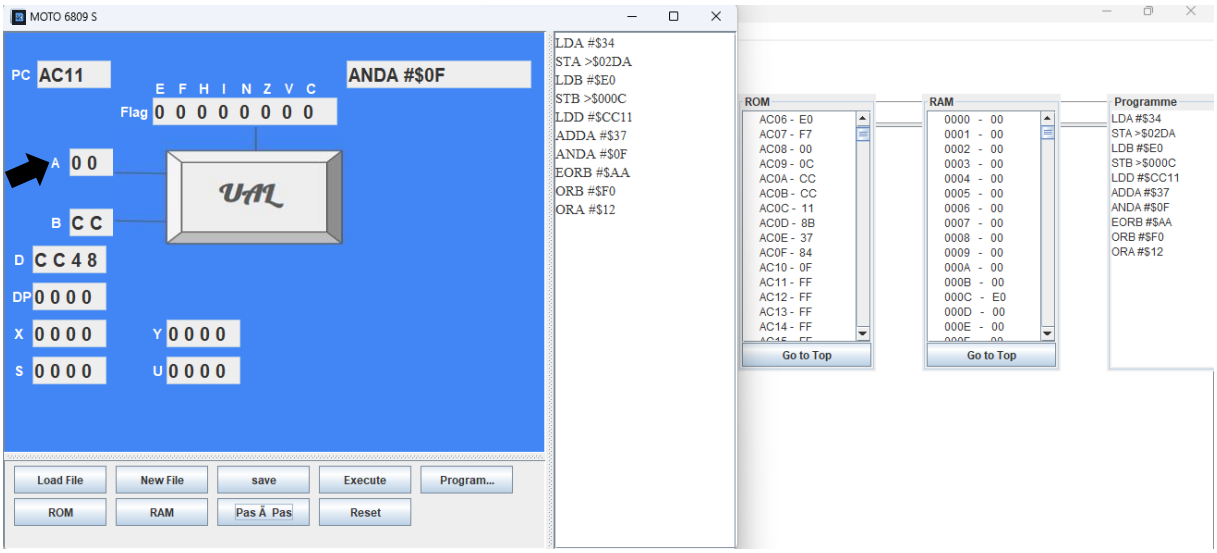

MOTO 6809 S
PC AC05
STA >$02DA
E F H I N Z V C
Flag 0 0 0 0 0 1 0 0
A 34
B 00
D 0034
DP 0000
X 0000
Y 0000
S 0000
U 0000
UAL
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$34
STA >$02DA
LDB #$E0
STB >$000C
LDD #$CC11
ADDA #$37
ANDA #$0F
EORB #$AA
ORB #$F0
ORA #$12
ROM
AC00 - 86
AC01 - 34
AC02 - B7
AC03 - 02
AC04 - DA
AC05 - FF
AC06 - FF
AC07 - FF
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
02D1 - 00
02D2 - 00
02D3 - 00
02D4 - 00
02D5 - 00
02D6 - 00
02D7 - 00
02D8 - 00
02D9 - 00
02DA - 34
02DB - 00
02DC - 00
02DD - 00
02DE - 00
02DF - 00
Go to Top
Programme
LDA #$34
STA >$02DA
LDB #$E0
STB >$000C
LDD #$CC11
ADDA #$37
ANDA #$0F
EORB #$AA
ORB #$F0
ORA #$12

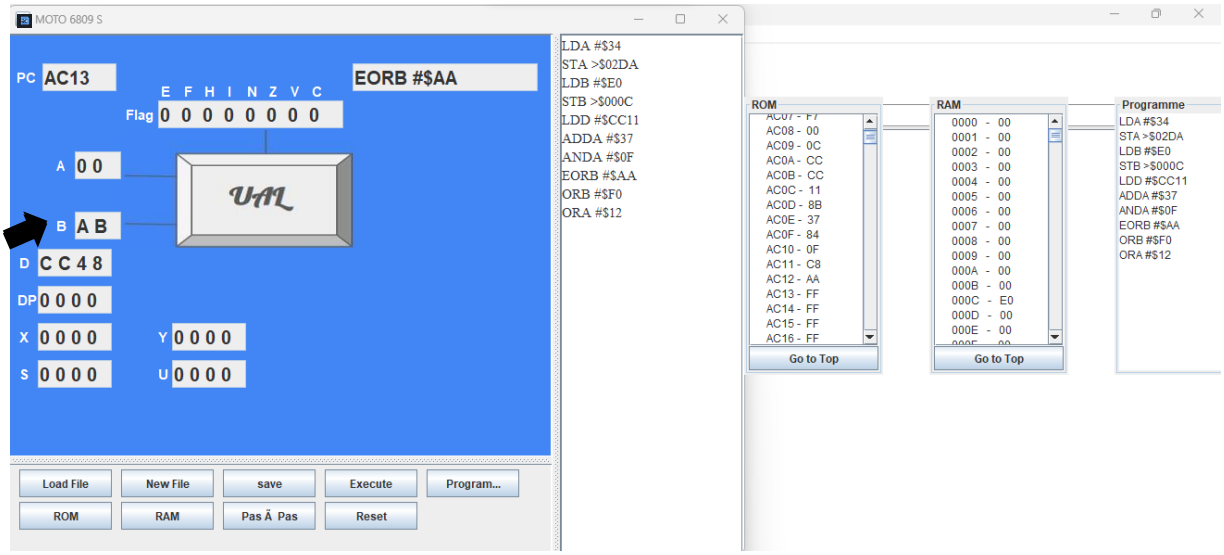

MOTO 6809 S
PC AC07
LDB #$E0
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 34
B E0
D E034
DP 0000
X 0000
Y 0000
S 0000
U 0000
UAL
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$34
STA >$02DA
LDB #$E0
STB >$000C
LDD #$CC11
ADDA #$37
ANDA #$0F
EORB #$AA
ORB #$F0
ORA #$12
ROM
AC00 - 86
AC01 - 34
AC02 - B7
AC03 - 02
AC04 - DA
AC05 - C6
AC06 - E0
AC07 - FF
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
LDA #$34
STA >$02DA
LDB #$E0
STB >$000C
LDD #$CC11
ADDA #$37
ANDA #$0F
EORB #$AA
ORB #$F0
ORA #$12

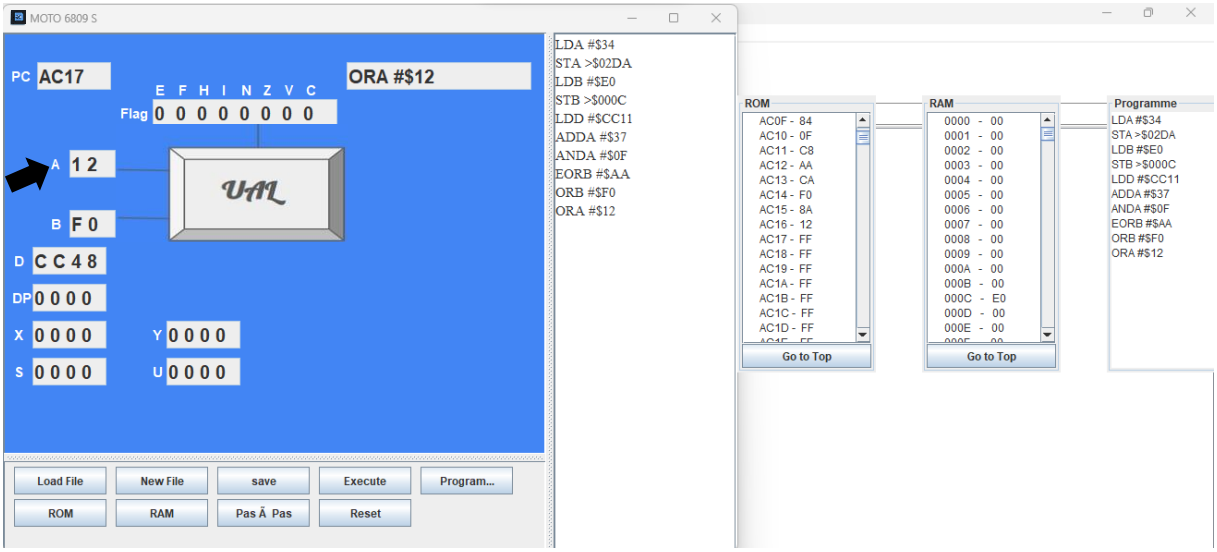

MOTO 6809 S
PC AC0A
STB >$000C
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 34
B E0
D E034
DP 0000
X 0000
Y 0000
S 0000
U 0000
UAL
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$34
STA >$02DA
LDB #$E0
STB >$000C
LDD #$CC11
ADDA #$37
ANDA #$0F
EORB #$AA
ORB #$F0
ORA #$12
ROM
AC00 - 86
AC01 - 34
AC02 - B7
AC03 - 02
AC04 - DA
AC05 - C6
AC06 - E0
AC07 - F7
AC08 - 00
AC09 - 0C
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - E0
000D - 00
000E - 00
Go to Top
Programme
LDA #$34
STA >$02DA
LDB #$E0
STB >$000C
LDD #$CC11
ADDA #$37
ANDA #$0F
EORB #$AA
ORB #$F0
ORA #$12


MOTO 6809 S
PC AC0D
LDD #$CC11
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 1 1
B C C
D C C 1 1
DP 0 0 0 0
X 0 0 0 0
Y 0 0 0 0
S 0 0 0 0
U 0 0 0 0
UAL
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$34
STA >$02DA
LDB #$E0
STB >$000C
LDD #$CC11
ADDA #$37
ANDA #$0F
EORB #$AA
ORB #$F0
ORA #$12
ROM
AC00 - 86
AC01 - 34
AC02 - B7
AC03 - 02
AC04 - DA
AC05 - C6
AC06 - E0
AC07 - F7
AC08 - 00
AC09 - 0C
AC0A - CC
AC0B - CC
AC0C - 11
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - E0
000D - 00
000E - 00
Go to Top
Programme
LDA #$34
STA >$02DA
LDB #$E0
STB >$000C
LDD #$CC11
ADDA #$37
ANDA #$0F
EORB #$AA
ORB #$F0
ORA #$12

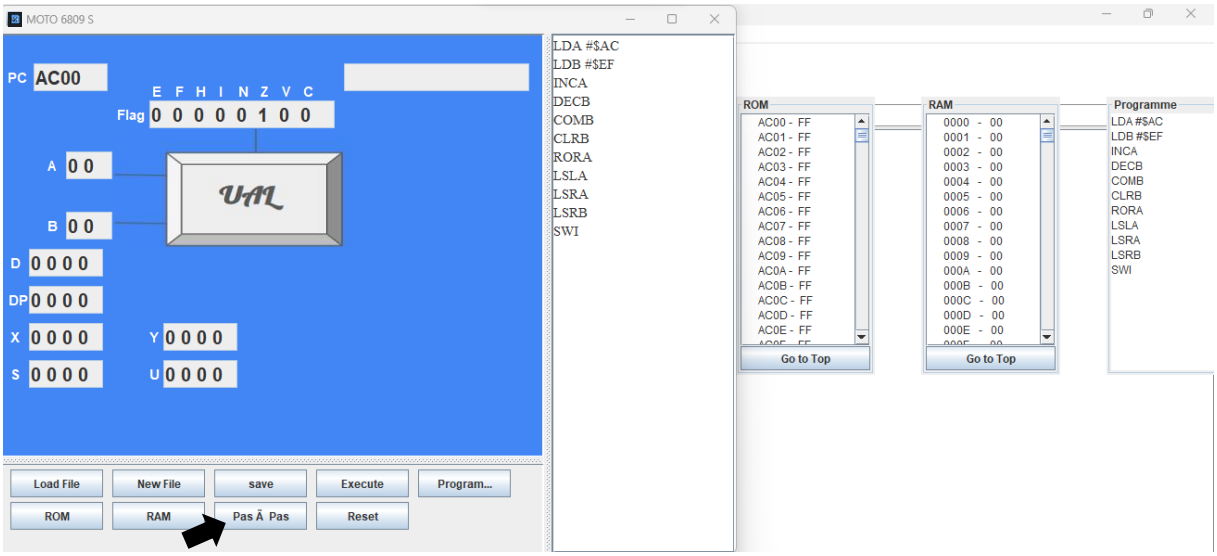

MOTO 6809 S
PC AC0F
ADDA #$37
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 4 8
B C C
D C C 4 8
DP 0 0 0 0
X 0 0 0 0
Y 0 0 0 0
S 0 0 0 0
U 0 0 0 0
UAL
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$34
STA >$02DA
LDB #$E0
STB >$000C
LDD #$CC11
ADDA #$37
ANDA #$0F
EORB #$AA
ORB #$F0
ORA #$12
ROM
AC04 - DA
AC05 - C6
AC06 - E0
AC07 - F7
AC08 - 00
AC09 - 0C
AC0A - CC
AC0B - CC
AC0C - 11
AC0D - 8B
AC0E - 37
AC0F - FF
AC10 - FF
AC11 - FF
AC12 - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - E0
000D - 00
000E - 00
Go to Top
Programme
LDA #$34
STA >$02DA
LDB #$E0
STB >$000C
LDD #$CC11
ADDA #$37
ANDA #$0F
EORB #$AA
ORB #$F0
ORA #$12

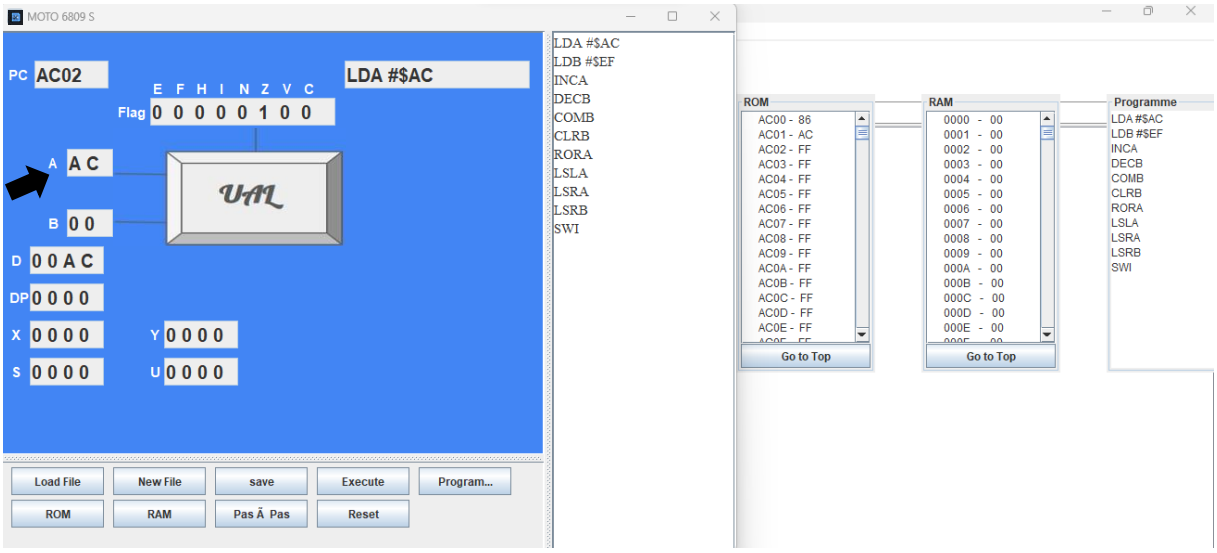

MOTO 6809 S
PC AC11
ANDA #$0F
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 0 0
B C C
D C C 4 8
DP 0 0 0 0
X 0 0 0 0
Y 0 0 0 0
S 0 0 0 0
U 0 0 0 0
UAL
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$34
STA >$02DA
LDB #$E0
STB >$000C
LDD #$CC11
ADDA #$37
ANDA #$0F
EORB #$AA
ORB #$F0
ORA #$12
ROM
AC06 - E0
AC07 - F7
AC08 - 00
AC09 - 0C
AC0A - CC
AC0B - CC
AC0C - 11
AC0D - 8B
AC0E - 37
AC0F - 84
AC10 - 0F
AC11 - FF
AC12 - FF
AC13 - FF
AC14 - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - E0
000D - 00
000E - 00
Go to Top
Programme
LDA #$34
STA >$02DA
LDB #$E0
STB >$000C
LDD #$CC11
ADDA #$37
ANDA #$0F
EORB #$AA
ORB #$F0
ORA #$12

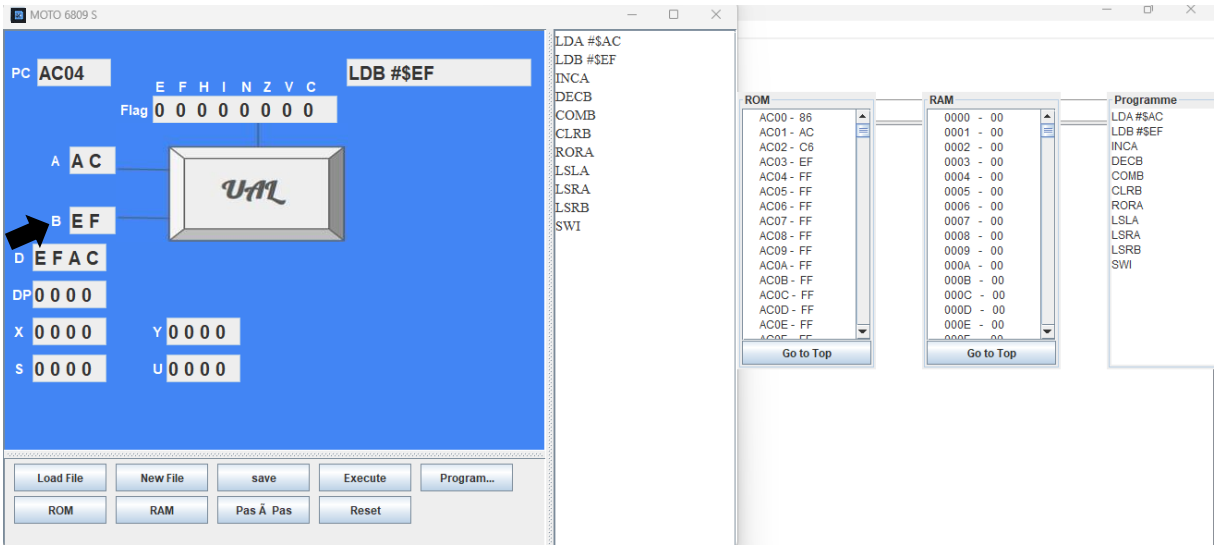


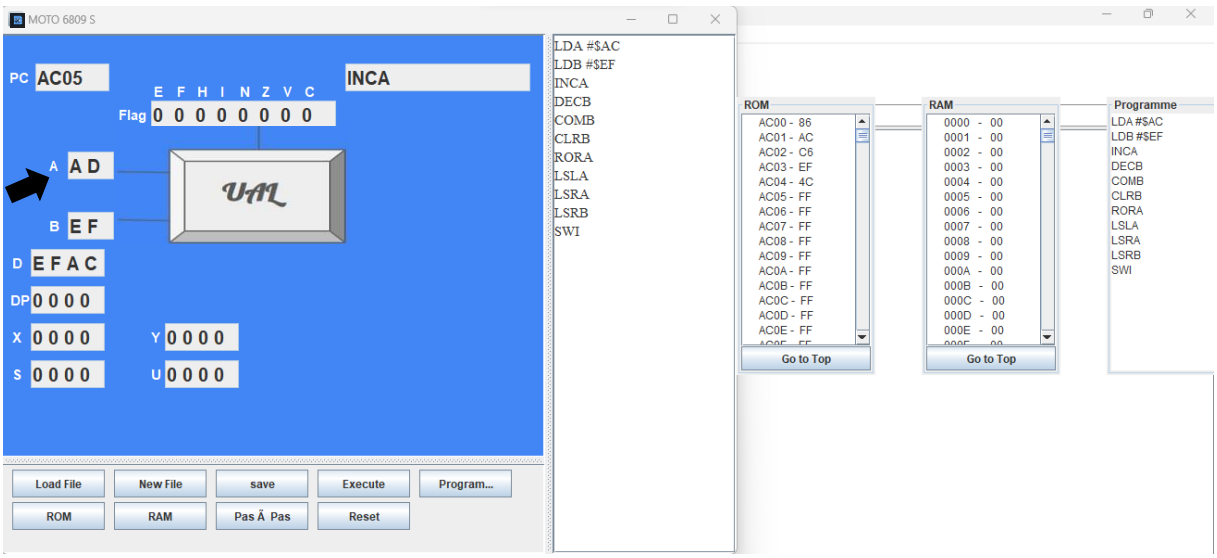


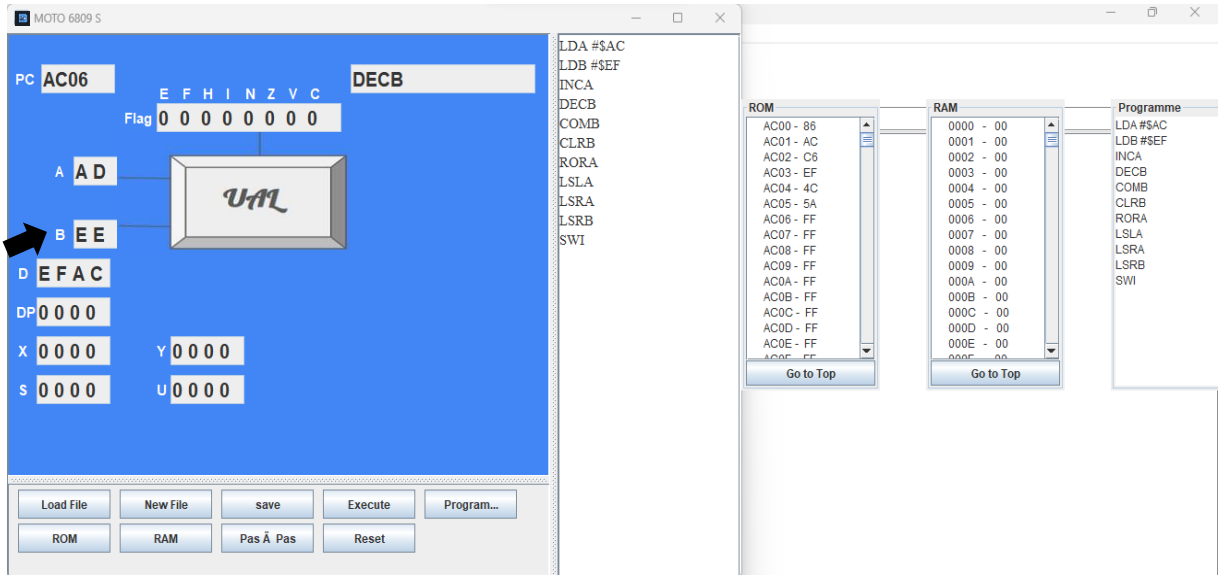
○ Exemple 3 : D’autres instructions :

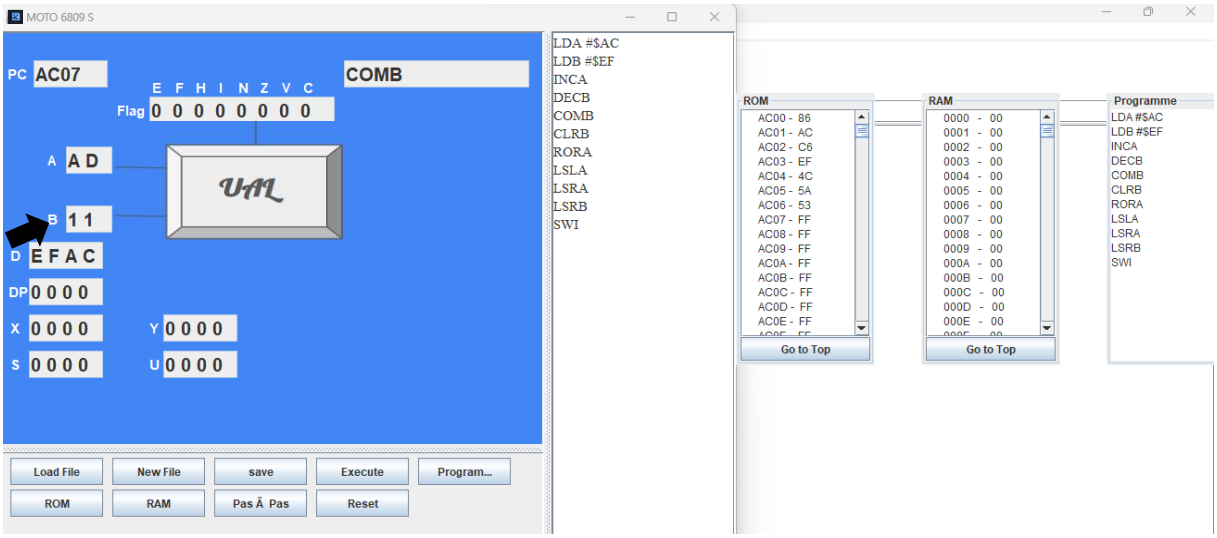


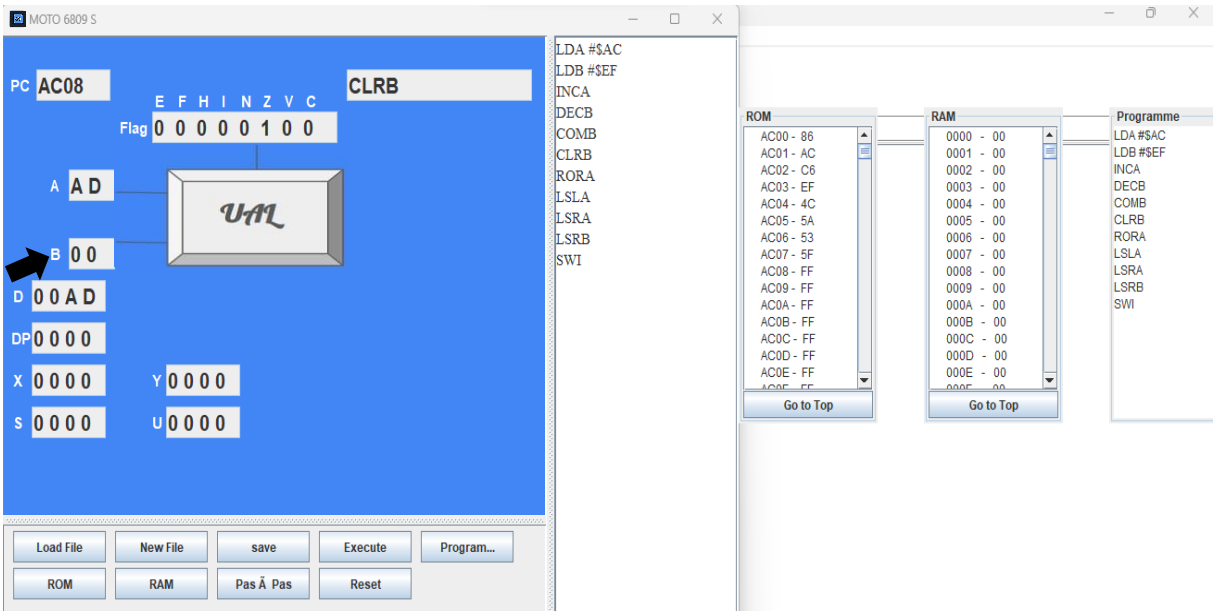

MOTO 6809 S
PC AC02
LDA #$AC
E F H I N Z V C
Flag 0 0 0 0 0 1 0 0
A AC
B 00
D 00AC
DP 0000
X 0000
Y 0000
S 0000
U 0000
UAL
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$AC
LDB #$EF
INCA
DECB
COMB
CLRB
RORA
LSLA
LSRA
LSRB
SWI
ROM
AC00 - 86
AC01 - AC
AC02 - FF
AC03 - FF
AC04 - FF
AC05 - FF
AC06 - FF
AC07 - FF
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
LDA #$AC
LDB #$EF
INCA
DECB
COMB
CLRB
RORA
LSLA
LSRA
LSRB
SWI

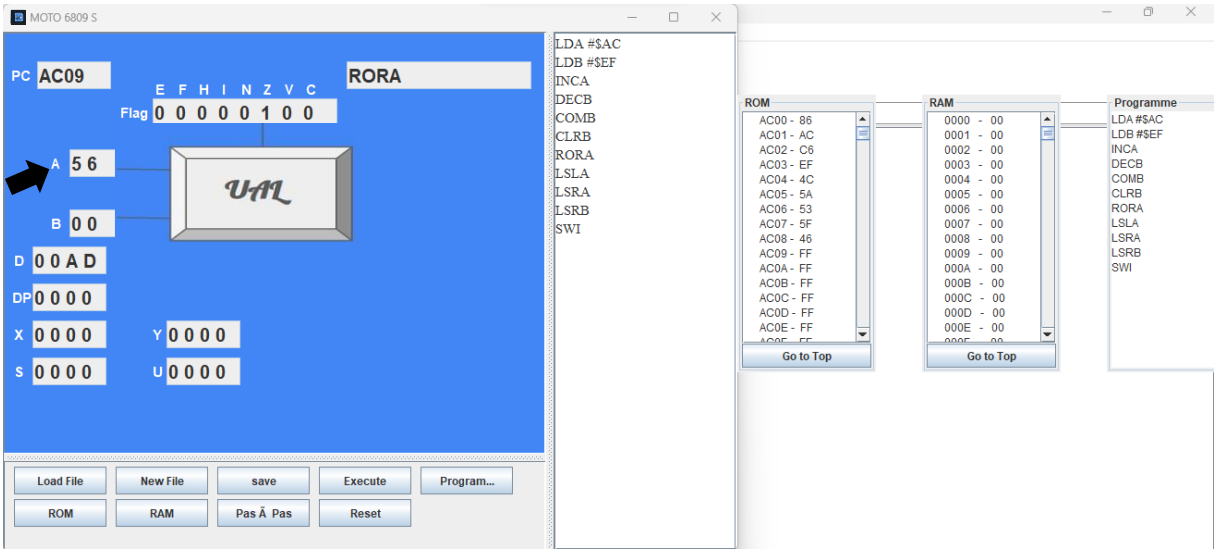

MOTO 6809 S
PC AC04
LDB #$EF
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A AC
B EF
D EFAC
DP 0000
X 0000
Y 0000
S 0000
U 0000
UAL
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$AC
LDB #$EF
INCA
DECB
COMB
CLRB
RORA
LSLA
LSRA
LSRB
SWI
ROM
AC00 - 86
AC01 - AC
AC02 - C6
AC03 - EF
AC04 - FF
AC05 - FF
AC06 - FF
AC07 - FF
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
LDA #$AC
LDB #$EF
INCA
DECB
COMB
CLRB
RORA
LSLA
LSRA
LSRB
SWI


MOTO 6809 S
PC AC05
INCA
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A AD
B EF
D EFAC
DP 0000
X 0000
Y 0000
S 0000
U 0000
UAL
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$AC
LDB #$EF
INCA
DECB
COMB
CLRB
RORA
LSLA
LSRA
LSRB
SWI
ROM
AC00 - 86
AC01 - AC
AC02 - C6
AC03 - EF
AC04 - 4C
AC05 - FF
AC06 - FF
AC07 - FF
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
LDA #$AC
LDB #$EF
INCA
DECB
COMB
CLRB
RORA
LSLA
LSRA
LSRB
SWI


MOTO 6809 S
PC AC06
DECB
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A AD
B EE
UAL
D EFAC
DP 0000
X 0000
Y 0000
S 0000
U 0000
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$AC
LDB #$EF
INCA
DECB
COMB
CLRB
RORA
LSLA
LSRA
LSRB
SWI
ROM
AC00 - 86
AC01 - AC
AC02 - C6
AC03 - EF
AC04 - 4C
AC05 - 5A
AC06 - FF
AC07 - FF
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
LDA #$AC
LDB #$EF
INCA
DECB
COMB
CLRB
RORA
LSLA
LSRA
LSRB
SWI


MOTO 6809 S
PC AC07
COMB
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A AD
B 11
UAL
D EFAC
DP 0000
X 0000
Y 0000
S 0000
U 0000
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$AC
LDB #$EF
INCA
DECB
COMB
CLRB
RORA
LSLA
LSRA
LSRB
SWI
ROM
AC00 - 86
AC01 - AC
AC02 - C6
AC03 - EF
AC04 - 4C
AC05 - 5A
AC06 - 53
AC07 - FF
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
LDA #$AC
LDB #$EF
INCA
DECB
COMB
CLRB
RORA
LSLA
LSRA
LSRB
SWI

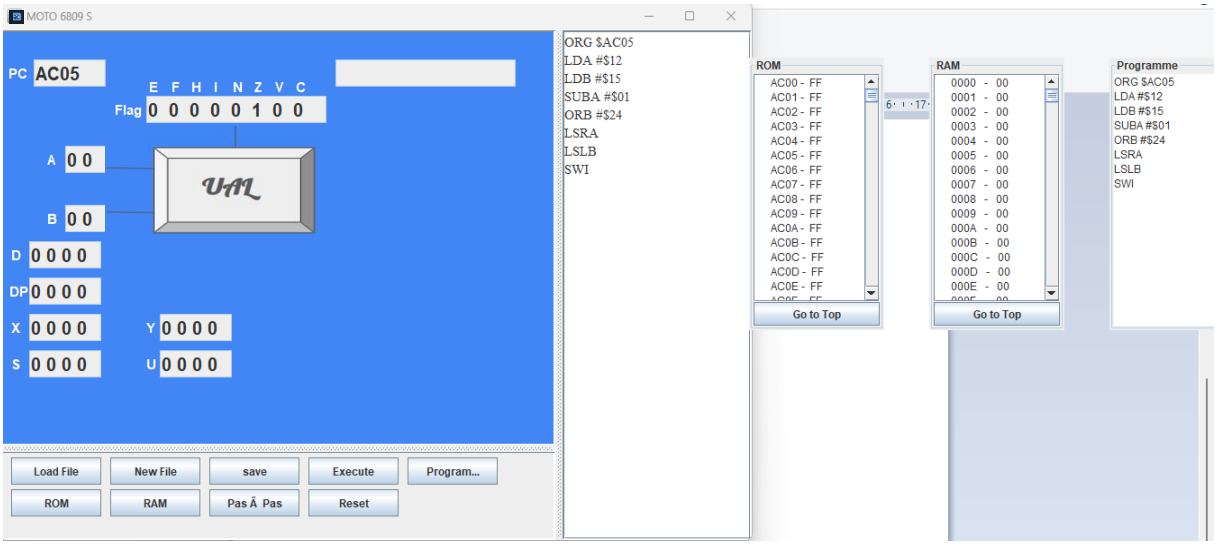

MOTO 6809 S
PC AC08
CLRB
E F H I N Z V C
Flag 0 0 0 0 0 1 0 0
A AD
B 00
UAL
D 00AD
DP 0000
X 0000
Y 0000
S 0000
U 0000
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
LDA #$AC
LDB #$EF
INCA
DECB
COMB
CLRB
RORA
LSLA
LSRA
LSRB
SWI
ROM
AC00 - 86
AC01 - AC
AC02 - C6
AC03 - EF
AC04 - 4C
AC05 - 5A
AC06 - 53
AC07 - 5F
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
LDA #$AC
LDB #$EF
INCA
DECB
COMB
CLRB
RORA
LSLA
LSRA
LSRB
SWI

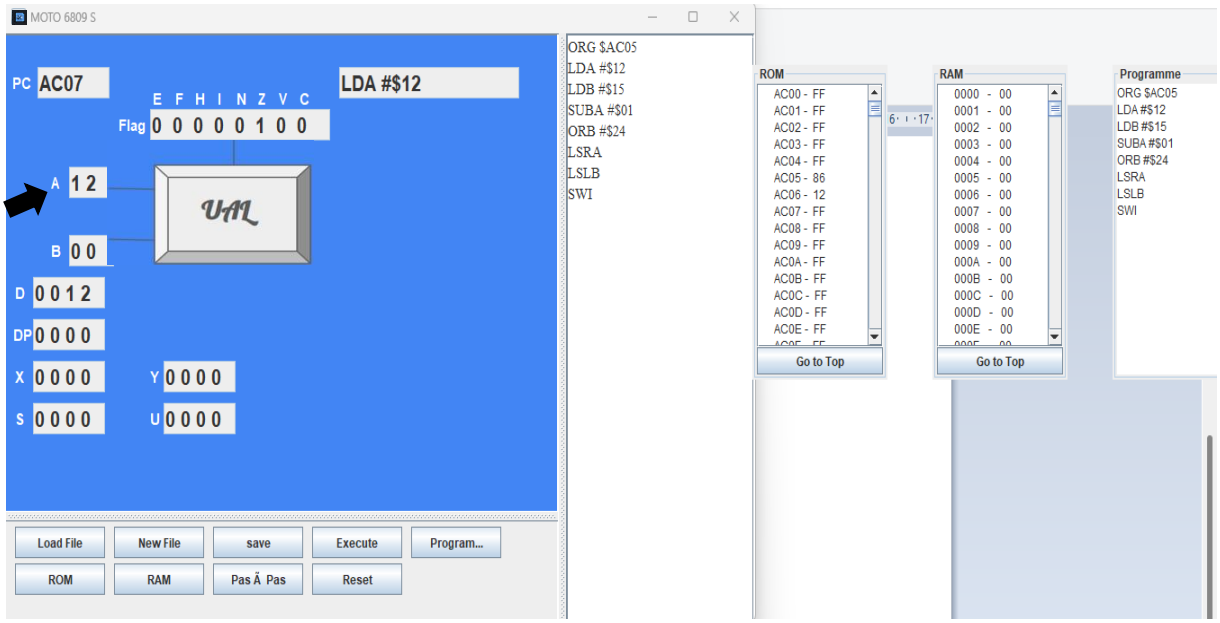


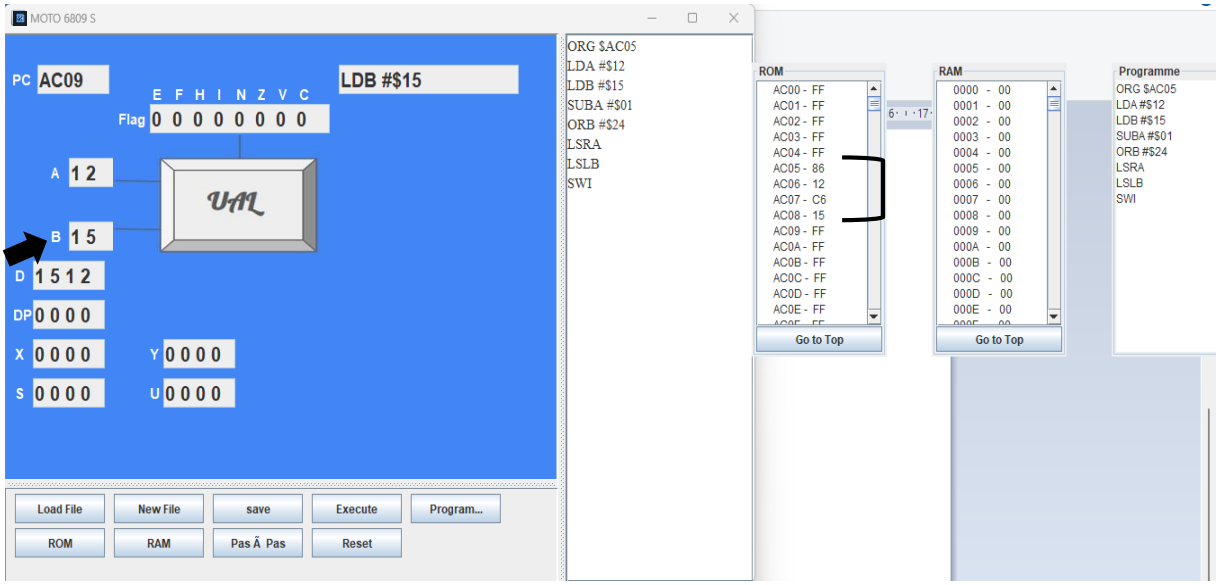
Un autre programme Assembleur :

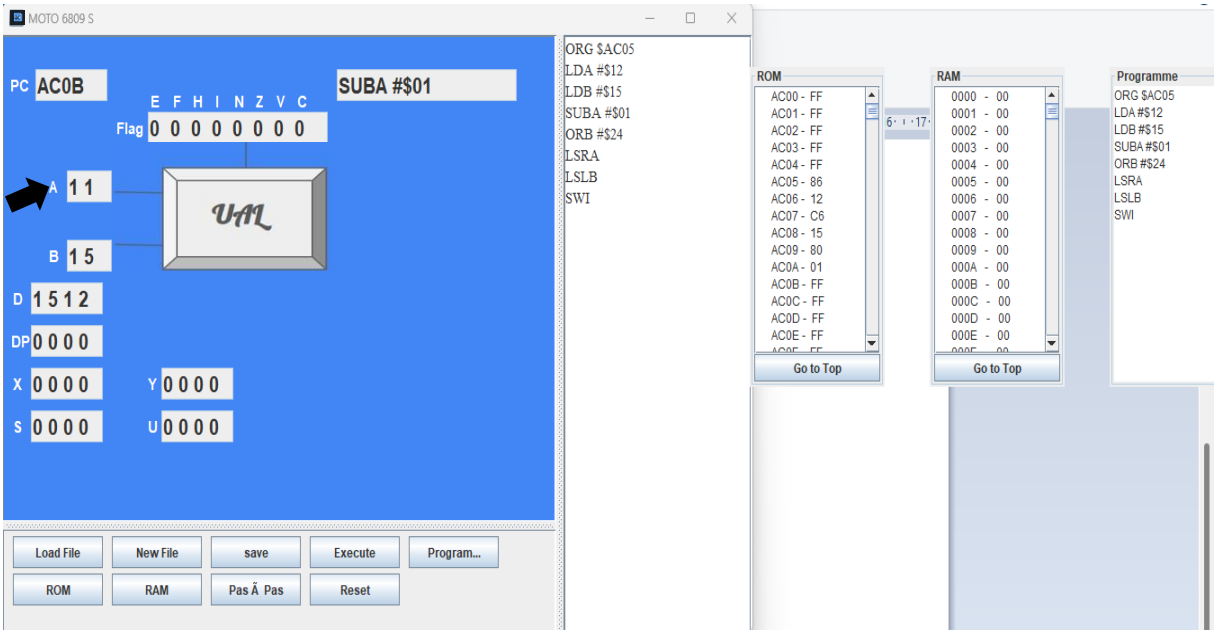
Remarque : l'instruction « **ORG** » est utilisée pour indiquer à l'assembleur où placer une section spécifique de code ou données en mémoire lors de la création du programme.

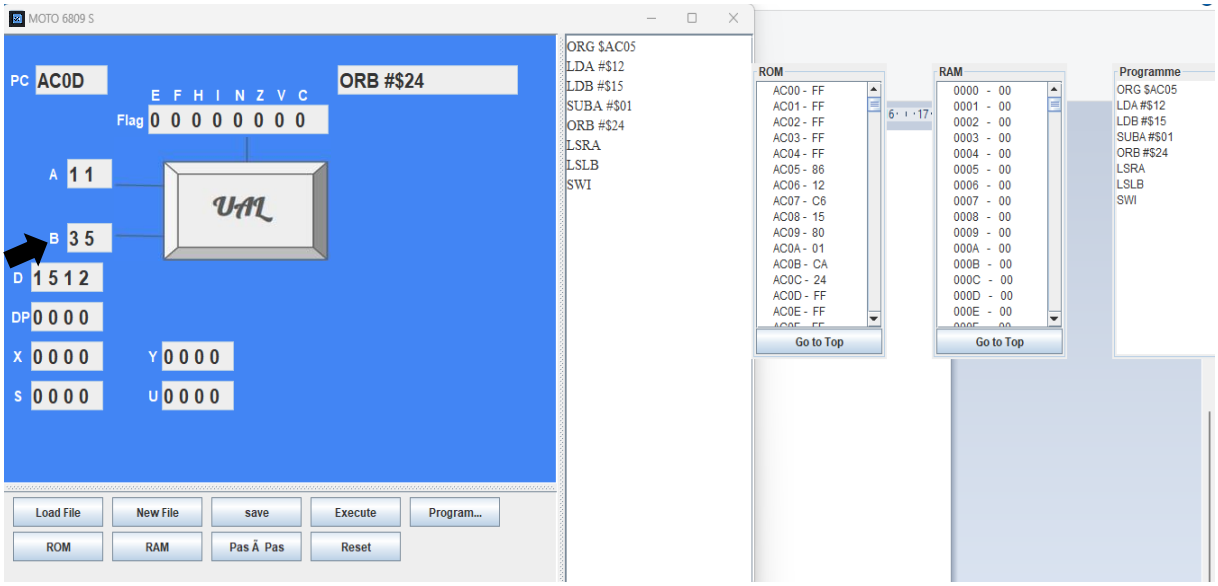
Cela n'occupe pas d'espace mémoire en soi, mais cela affecte l'emplacement où les éléments du programme seront stockés en mémoire ROM lors de son exécution.

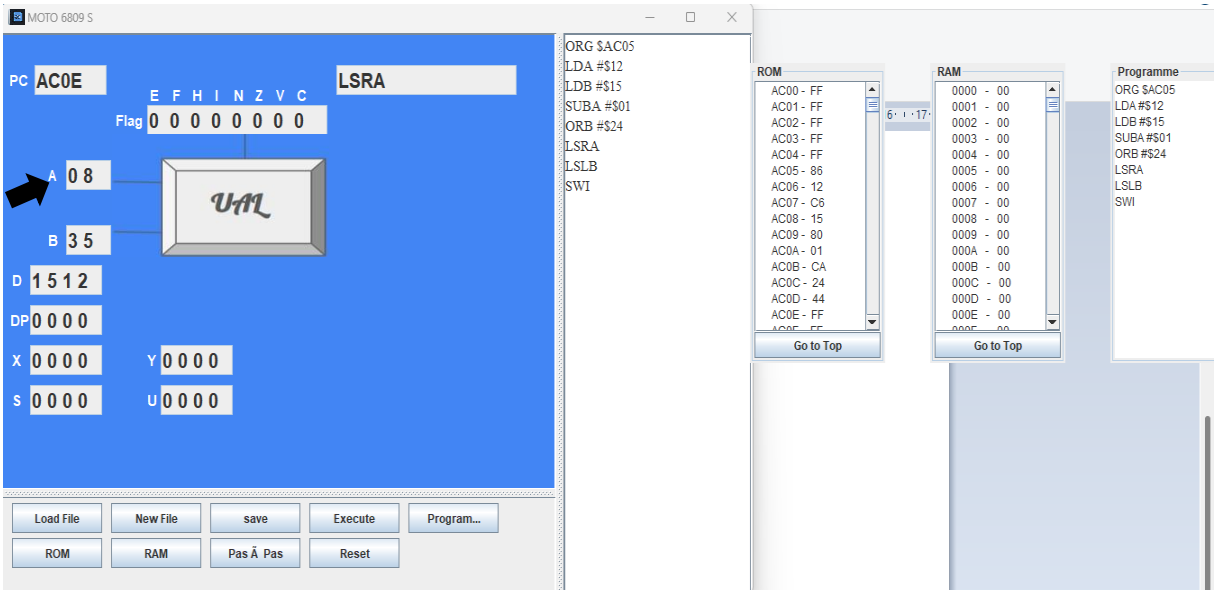


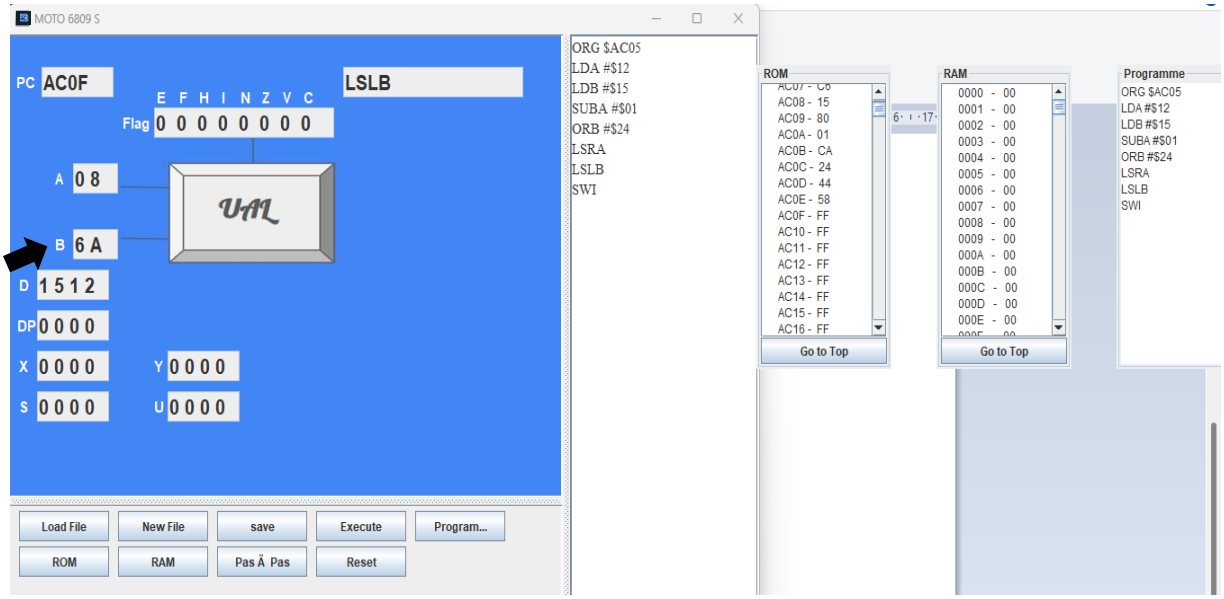

MOTO 6809 S
PC AC07
LDA #$12
E F H I N Z V C
Flag 0 0 0 0 0 1 0 0
A 1 2
B 0 0
UAL
D 0 0 1 2
DP 0 0 0 0
X 0 0 0 0
Y 0 0 0 0
S 0 0 0 0
U 0 0 0 0
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI
ROM
AC00 - FF
AC01 - FF
AC02 - FF
AC03 - FF
AC04 - FF
AC05 - 86
AC06 - 12
AC07 - FF
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI

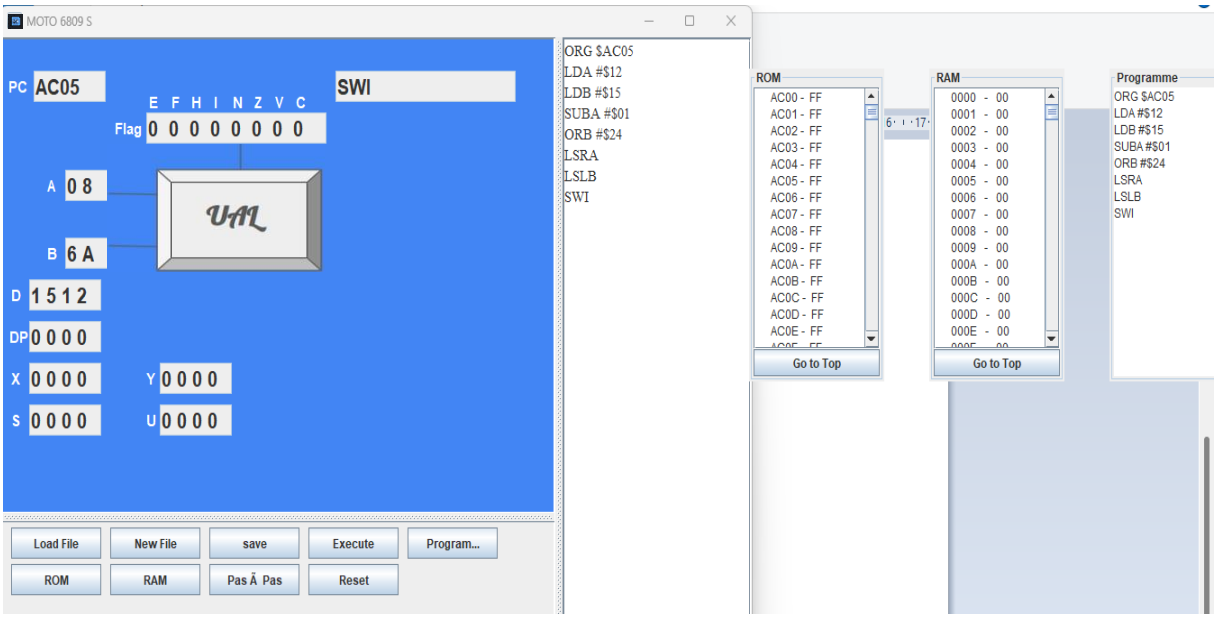

MOTO 6809 S
PC AC09
LDB #$15
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 1 2
B 1 5
UAL
D 1 5 1 2
DP 0 0 0 0
X 0 0 0 0
Y 0 0 0 0
S 0 0 0 0
U 0 0 0 0
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI
ROM
AC00 - FF
AC01 - FF
AC02 - FF
AC03 - FF
AC04 - FF
AC05 - 86
AC06 - 12
AC07 - C6
AC08 - 15
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI


MOTO 6809 S
PC AC0B
SUBA #$01
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 1 1
B 1 5
UAL
D 1 5 1 2
DP 0 0 0 0
X 0 0 0 0
Y 0 0 0 0
S 0 0 0 0
U 0 0 0 0
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI
ROM
AC00 - FF
AC01 - FF
AC02 - FF
AC03 - FF
AC04 - FF
AC05 - 86
AC06 - 12
AC07 - C6
AC08 - 15
AC09 - 80
AC0A - 01
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI


MOTO 6809 S
PC AC0D
ORB #$24
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 1 1
B 3 5
UAL
D 1 5 1 2
DP 0 0 0 0
X 0 0 0 0
Y 0 0 0 0
S 0 0 0 0
U 0 0 0 0
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI
ROM
AC00 - FF
AC01 - FF
AC02 - FF
AC03 - FF
AC04 - FF
AC05 - 86
AC06 - 12
AC07 - C6
AC08 - 15
AC09 - 80
AC0A - 01
AC0B - CA
AC0C - 24
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset


MOTO 6809 S
PC AC0E
LSRA
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 0 8
B 3 5
UAL
D 1 5 1 2
DP 0 0 0 0
X 0 0 0 0
Y 0 0 0 0
S 0 0 0 0
U 0 0 0 0
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI
ROM
AC00 - FF
AC01 - FF
AC02 - FF
AC03 - FF
AC04 - FF
AC05 - 86
AC06 - 12
AC07 - C6
AC08 - 15
AC09 - 80
AC0A - 01
AC0B - CA
AC0C - 24
AC0D - 44
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset


MOTO 6809 S
PC AC0F
LSLB
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 0 8
B 6 A
UAL
D 1 5 1 2
DP 0 0 0 0
X 0 0 0 0
Y 0 0 0 0
S 0 0 0 0
U 0 0 0 0
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI
ROM
AC08 - 15
AC09 - 80
AC0A - 01
AC0B - CA
AC0C - 24
AC0D - 44
AC0E - 58
AC0F - FF
AC10 - FF
AC11 - FF
AC12 - FF
AC13 - FF
AC14 - FF
AC15 - FF
AC16 - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset


MOTO 6809 S
PC AC05
SWI
E F H I N Z V C
Flag 0 0 0 0 0 0 0 0
A 0 8
B 6 A
UAL
D 1 5 1 2
DP 0 0 0 0
X 0 0 0 0
Y 0 0 0 0
S 0 0 0 0
U 0 0 0 0
Load File
New File
save
Execute
Program...
ROM
RAM
Pas Ã Pas
Reset
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI
ROM
AC00 - FF
AC01 - FF
AC02 - FF
AC03 - FF
AC04 - FF
AC05 - FF
AC06 - FF
AC07 - FF
AC08 - FF
AC09 - FF
AC0A - FF
AC0B - FF
AC0C - FF
AC0D - FF
AC0E - FF
Go to Top
RAM
0000 - 00
0001 - 00
0002 - 00
0003 - 00
0004 - 00
0005 - 00
0006 - 00
0007 - 00
0008 - 00
0009 - 00
000A - 00
000B - 00
000C - 00
000D - 00
000E - 00
Go to Top
Programme
ORG $AC05
LDA #$12
LDB #$15
SUBA #$01
ORB #$24
LSRA
LSLB
SWI