Module 4: Langage assembleur (suite 2)

1 Introduction

Dans la leçon précédente, nous avons présenté certaines instructions du langage assembleur à savoir les instructions relatives aux ruptures de séquence ou branchements, les décalages et rotations, et les instructions d'entrées/sorties.

Dans cette leçon, nous allons toujours continuer avec les instructions du langage assembleur. Plus particulièrement, nous allons présenter les instructions d'interruptions et de synchronisation externe. Nous allons présenter à la fin de cette leçon, un outil visuel simple permettant l'écriture et l'exécution du code assembleur.

2 Instructions d'interruption

À l'instar d'un appel de procédure, une interruption exécute une routine d'interruption par adressage indirect via un vecteur d'interruption 32 bits se trouvant en mémoire. En plus de l'adresse de retour, elle range les FLAGS sur la pile (l'instruction PUSHF par exemple). Il y a trois sortes d'instructions : deux instructions d'appel et une de retour.

— INT : INTerrupt, quand l'instruction INT est exécutée, le processeur met sur la pile les FLAGS, CS et IP, met TF=IF=0 et fait pointer CS :IP à l'adresse de départ de la routine d'interruption. il y a 256 vecteurs d'interruption localisés dans les cases mémoires les plus basses occupant un total de 1 Koctet (0 à 3FFH).

TF *(bit 8) Trap Flag (Drapeau de trappe) : ce drapeau permet le débogage en mode pas à pas, c'est-à-dire instruction par instruction.*
IF *(bit 9) Interrupt Flag (Drapeau d'interruption) : Ce drapeau contrôle la façon dont le processeur répond aux requêtes d'interruptions masquables (c'est-à-dire désactivables). Lorsqu'il est activé, le processeur peut répondre à toutes les interruptions, dans le cas contraire (drapeau IF désactivé), le processeur ne pourra répondre qu'aux interruptions non masquables*[1].

— IRET : Interrupt RETurn, c'est la dernière instruction de la routine d'interruption. IRET récupère les 3 mots au haut de la pile et les charge respectivement dans IP, CS et les FLAGS. Cette instruction est équivalente à POP IP, POP CS, et POP F.

```
1    MOV AL, 13h
2    MOV AH, 0
3    INT 10h      ; Interruption.
4    ...          ; Routine d'interruption
5    ...
6    IRET         ; Retour
```

— IRETD : Interrupt RETurn with Double word, cette instruction dépile les doubles mots EIP, puis CS (le mot du poids fort est abandonné) et enfin EFLAGS (E pour extended, 32 bits).
— INTO : Interrupt if Overflow, c'est l'interruption qui est activée lorsque OF=1.

2.1 Instructions de synchronisation externe

L'instruction HLT (HALT), maintient le processeur dans un état d'attente d'un reset ou d'une interruption externe (avec IF = 1).

L'instruction WAIT met également le processeur en attente, répond aux interruptions, mais s'occupe à tester l'entrée TEST toutes les 5 périodes d'horloge : si elle est active, le processeur exécute l'instruction suivante à WAIT. Le but est une synchronisation avec un périphérique externe.

L'instruction LOCK est un préfixe pour toute instruction permettant le verrouillage du bus (vis-à-vis des autres processeurs) pendant son exécution. Elle est donc très utile dans les systèmes multi-processeurs.

L'instruction NOP (No OPeration) ne fait rien. On peut s'en servir dans une boucle de temporisation par exemple, ou bien pour pouvoir supprimer, dans un programme, une instruction sans avoir besoin de réassembler, ou bien encore pour arrêter le pas à pas dans un débogage.

3 Extensions possibles

Les instructions présentées dans ce module permettent d'écrire des petite programmes en langage assembleur, mais l'on peut étendre le jeu d'instruction pour le rendre plus puissant : opérations sur des nombres non signés ou flottants, gestion de la pile et appels aux fonctions, prise en charge avancée des interruptions, etc. Nous allons présenter quelques instructions intéressantes supplémentaires, qui ne sont pas nécessaires en première lecture pour comprendre le langage assembleur. Le tableau suivant résume quelques instructions avancées [2] :

2. Pour plus de détails sur ces instructions, veuillez vous référer au livre d'architecture de l'ordinateur d'Emmanuel Lazard.

TABLE 1 – Instructions supplémentaires possibles.

Mnémonique	Description
LDBU AX, (DX) LDHU AX, (DX)	Le chargement de 1 ou 2 octets depuis la mémoire dans un registre provoque une extension de signe. Il est parfois important de l'éviter et de remplacer les octets de poids fort (non chargés) du registre par 0 ; c'est le cas lorsque l'on travaille avec des caractères ou des entiers non signés (LDHU, LoaD Byte/Half Unsigned).
CALL adr RET	L'instruction CALL provoque un appel de fonction à l'adresse Adr en empilant le registre PC pour sauvegarder l'adresse de retour. La dernière instruction de la fonction, RET (RETurn), récupère la valeur en haut de la pile et la met dans le registre PC pour reprendre le programme après l'appel.
TRAP RETI	TRAP provoque une interruption logicielle qui déroute l'exécution à une adresse précise indiquée dans le processeur. RETI (RETurn from Interrupt) termine le traitement de l'interruption et reprend le cours normal du programme.
RST	RST (ReSeT) provoque un redémarrage du processeur, qui arrête tous les traitements en cours.
FMOV fX,fY FMVI fX,#f LDF fX,(rY) STF (rX),fY	Ces instructions organisent les transferts de données flottantes entre registres (FMOV), depuis ou vers la mémoire (LDF/STF), d'une donnée flottante immédiate (FMVI).
FADD fX,fY,fZ FSUB fX,fY,fZ FMUL fX,fY,fZ FDIV fX,fY,fZ	Ces instructions effectuent une opération flottante simple sur des valeurs situées dans les registres.

4 Outils de programmation en assembleur

Il existe plusieurs outils mis à la disposition des programmeurs pour développer des programmes en assembleur.

Pour des raisons de simplicité, nous allons utiliser dans ce cours l'émulateur 8086 que vous pouvez télécharger à l'adresse : `https://softfamous.com/emu8086/` comme le montre la figure 1

EMU8086
8086 MICROPROCESSOR EMULATOR
INTEGRATED ASSEMBLER AND DISASSEMBLER

version 5.0

DOWNLOAD
(40 day trial)

FOR ALL YOUR INQUERIES AND QUESTIONS:
info@emu8086.com

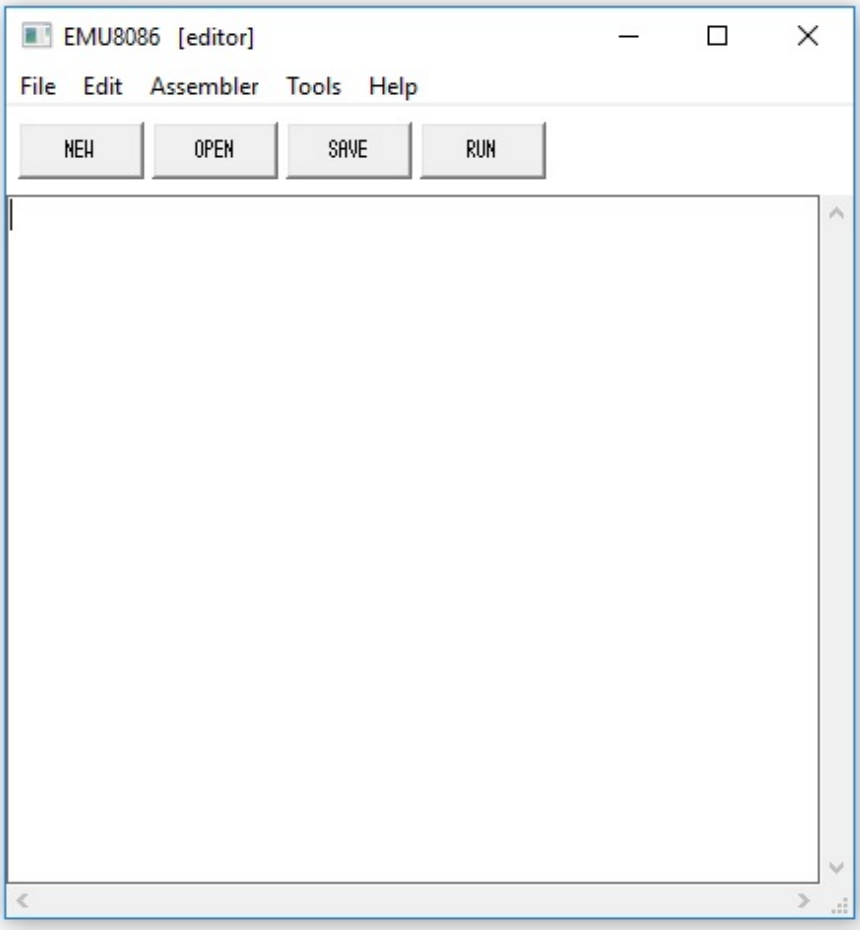

FIGURE 1 – Émulateur 8086

Dans l'émulateur 8086, nous pouvons créer des nouveaux fichier assembleur (.asm), ou bien ouvrir des fichiers existants (.asm) pour exécution. Nous pouvons également sauvegarder les nouveaux fichier que vous venez de créer.

Pour écrire du code assembleur, nous utilisons le bloc de texte comme le montre la figure suivante :

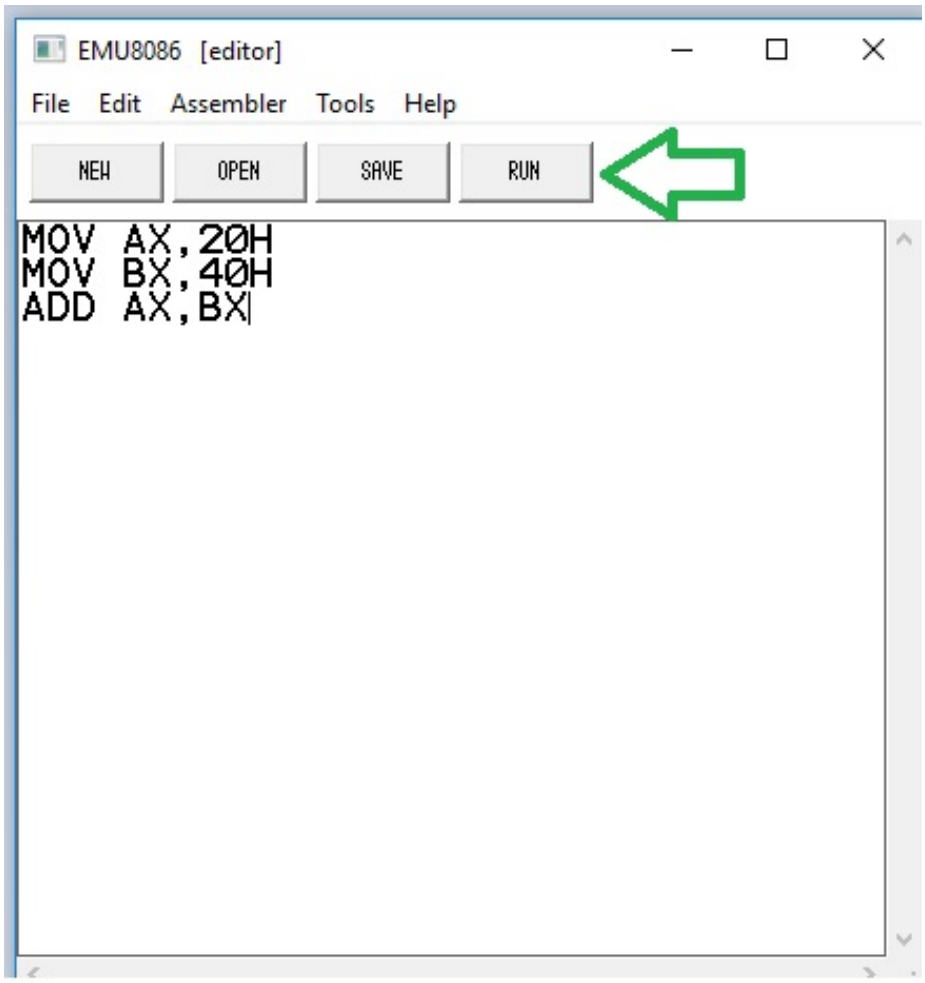

FIGURE 2 – Code assembleur

Une fois le code est fini, nous pouvons donc l'exécuter en cliquant sur le bouton "RUN". Une nouvelle fenêtre s'affiche comme illustré dans la figure suivante :

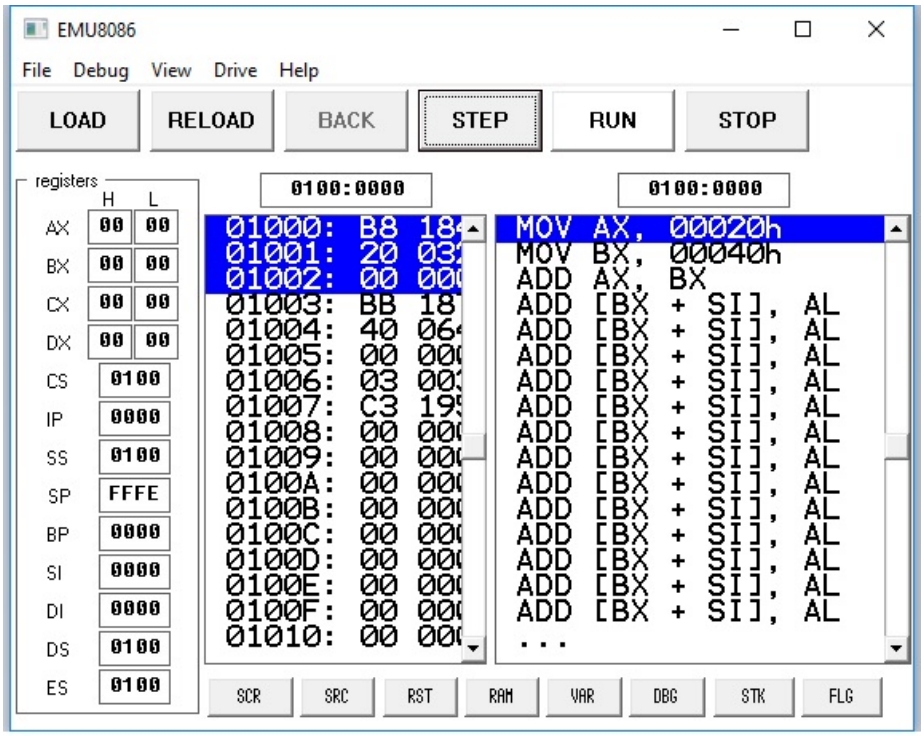

FIGURE 3 – Fenêtre d'exécution du code assembleur

Dans cette fenêtre, les registres sont affichés dans la partie gauche, et les instructions dans la partie droite.

La méthode la plus appropriée et recommandée pour exécuter des programmes en assembleur c'est via le débogage. Le débogage permet d'exécuter le code assmebleur instruction par instruction ce qui permet une meilleur comprhénesion du processus d'exécution et une meilleur interprétation des résultats. La fonction du débogage est disponible dans l'émulateur 8086 dans le menu "Debug". Pour exécuter le code assembleur pas à pas, nous choisissons l'option "single step" ou bien la fonction "F8" sur windows. Comme le montre l'exécution de l'exemple précédent dans les figures suivantes :

Au début, tous les registres sont à 0, i.e. ils ne contiennent aucune valeur. Le pointeur d'instruction IP est également à 0000. Lorsque on commence le débogage, la première instruction met la valeur 20 dans le registre AX (MOV AX, 20H). Le résultat d'exécution est montré dans la figure 4 :

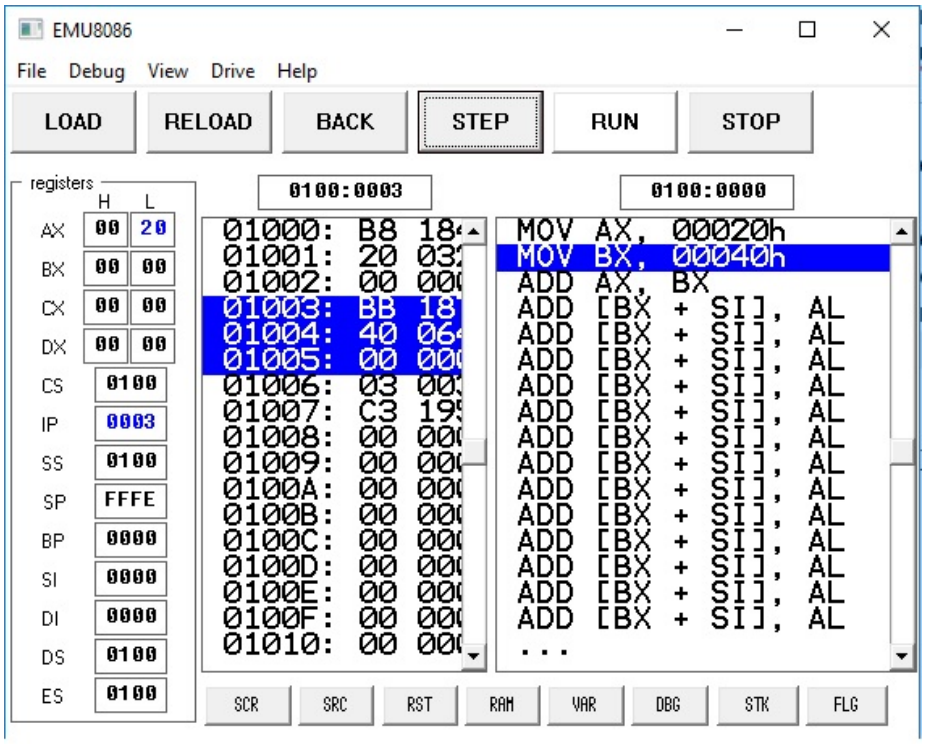

FIGURE 4 – Étape 1 d'exécution du code assembleur

Comme le montre la figure 4, le registre AX reçoit la valeur 20, et le pointeur d'instruction IP pointe sur la prochaine instruction à exécuter qui est située à l'adresse 0003.

Lorsque on continue la débogage, la prochaine instruction à exécuter est (MOV BX, 40H). L'exécution de cette instruction donne le résultat suivant :

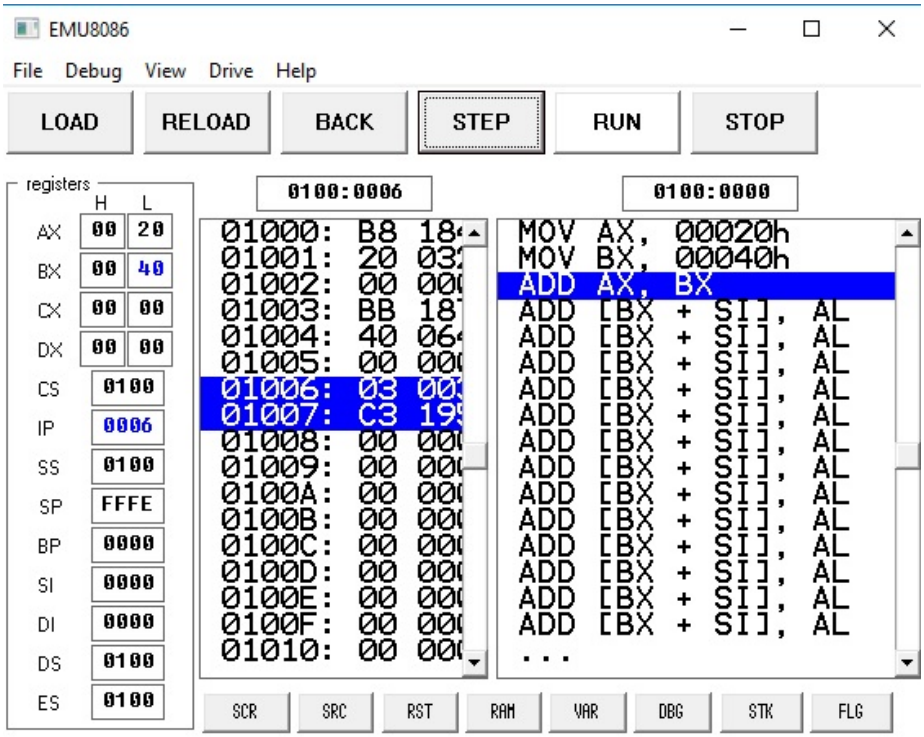

FIGURE 5 – Étape 2 d'exécution du code assembleur

Comme le montre la figure 5, le registre BX reçoit la valeur 40, et le pointeur d'instruction IP pointe sur la prochaine instruction à exécuter qui est située à l'adresse 0006 (IP est incrémenté de 3).

L'étape finale consiste donc à additionner les deux valeurs et mettre le résultat dans le registre AX. L'exécution de cette étape donne le résultat suivant :

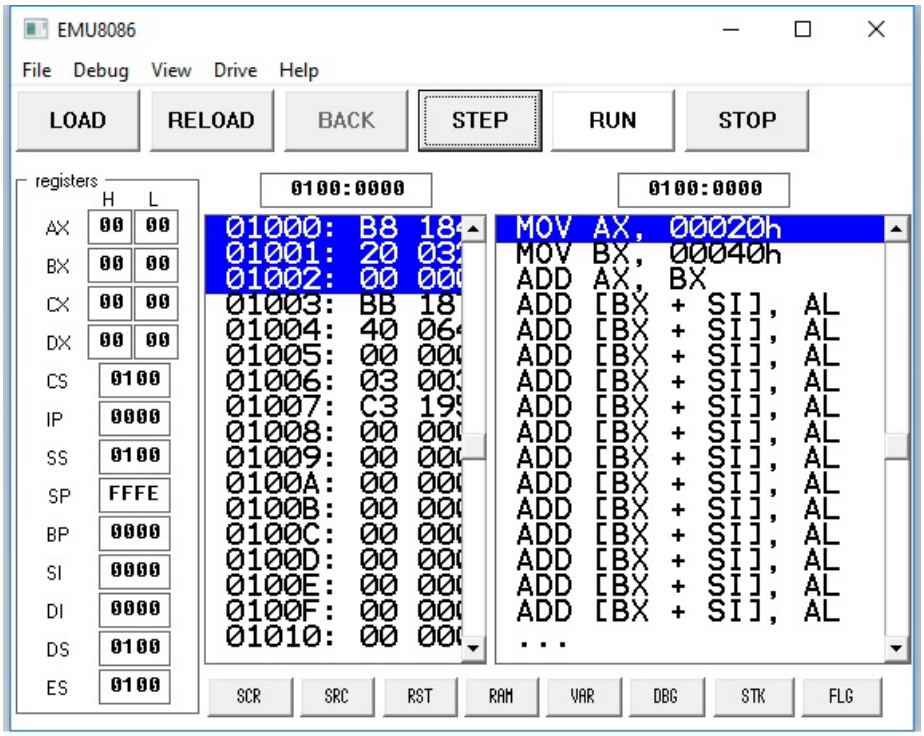

FIGURE 6 – Étape 3 d'exécution du code assembleur

Le résultat de l'opération d'addition, comme montré dans la figure 6, donne 60 (20 + 40) est stocké dans le registre AX. L'ancienne valeur du registre AX (20) a été remplacée par la nouvelle valeur (60).

En conclusion, cette outil offre une manipulation intuitive et conviviale pour s'amuser avec du code assembleur.

Il est recommandé d'utiliser cet outil dans le cadre de ce cours et dans tous les travaux notés en lien avec l'assembleur.