L1 I.EEEA
Architecture des ordinateurs
TP 1 : Premiers pas en assembleur 8086 avec TASM

Introduction

Les TP d'Architecture des ordinateurs visent à apprendre la programmation assembleur pour les microprocesseurs Intel 8086, premiers processeurs de la famille x86. L'objet de ce premier TP est de présenter et configurer l'environnement de développement et d'écrire, assembler et exécuter vos premiers programmes.

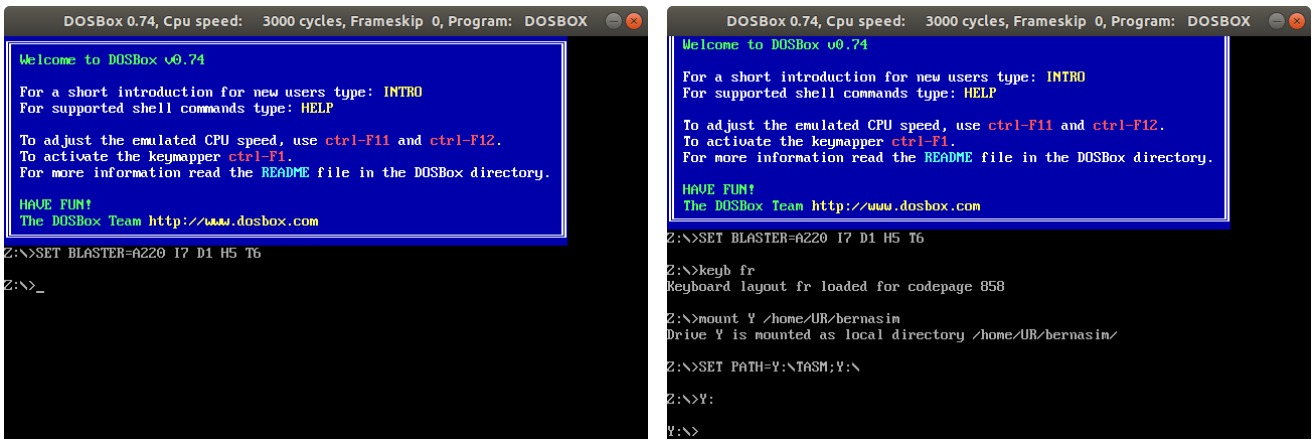
Configuration de l'environnement

L'émulateur DOSBox

Les ordinateurs d'aujourd'hui utilisent des microprocesseurs plus complexes que le processeur 8086. Donc pour ces TP, nous avons besoin de simuler une architecture de processeur Intel x86 sur laquelle exécuter nos programmes. Pour cela, nous allons utiliser l'émulateur DOSBox. Il s'agit d'un émulateur de machine recréant un environnement MS-DOS, et pouvant exécuter des programmes et des jeux vidéo développés autrefois pour ce système (source Wikipédia : `https://fr.wikipedia.org/wiki/DOSBox`). Cet émulateur est libre, gratuit et multiplateforme, c'est-à-dire qu'il fonctionne sur tous systèmes d'exploitation, dont Windows et Linux/Ubuntu.

▶ Démarrez DOSBox en ouvrant la grille d'applications (icône en bas de la barre d'application, le *dock*) et en saisissant les premières lettres de « dosbox » dans la barre de recherche. La fenêtre de la figure 1a doit apparaître.

Entre autres fonctionnalités, DOSBox permet de « monter » un lecteur virtuel à partir d'un répertoire de notre choix. Ce lecteur apparaîtra sous DOSBox sous la forme d'une unité de stockage (e.g. `C:\>`) dans laquelle le contenu du répertoire sera rendu accessible. Nous allons utiliser cette fonctionnalité pour accéder au répertoire personnel (*home*) d'Ubuntu à partir de DOSBox.



(a) (b)

FIGURE 1 – Émulateur DOSBox

▶ Ouvrez le document « Tutoriel configuration TASM » disponible sur UniversiTICE et suivez les instructions. A l'issue de ces manipulations vous devriez obtenir le résultat de la figure 1b.

Pour utiliser DOSBox, quelques raccourcis clavier vous seront utiles :

— `ctrl+F9` pour fermer la fenêtre DOSBox

— `ctrl+F10` pour récupérer la souris dans le système hôte (votre système d'exploitation à partir duquel vous exécutez DOSBox) si elle a été capturée par la fenêtre DOSBox

— `ctrl+F4` pour re-synchroniser le contenu des répertoires et le lecteur correspondant sous DOSBox

▶ Testez ces raccourcis clavier pour vous familiariser avec eux.

L'environnement MS-DOS

DOS était le système d'exploitation (OS pour *Operating System*) le plus utilisé sur PC jusqu'au début des années 90. MS-DOS en est la version commercialisée par Microsoft, à la base de leurs OS jusqu'à la création de Windows 95. Il est resté présent sur les PC Windows jusqu'au début des années 2000. Il s'agit d'un OS en ligne de commande et son rôle est d'interpréter les commandes saisies au clavier par l'utilisateur, notamment pour :

— la gestion des fichiers et des répertoires

— la mise à jour des disques

— la configuration du matériel

— l'optimisation de la mémoire

— l'exécution des programmes

Les commandes utilisateurs sont à saisir à l'invite (ou *prompt*) dans une console, c'est-à-dire dans le cas de MS-DOS après la lettre d'unité de stockage suivie de `:\>` (cf. figure 2). Voici une liste de commandes DOS dont vous aurez besoin en TP :

`dir`	:	liste le contenu d'un répertoire
`cd`	:	change de répertoire
`cd ..`	:	change de répertoire vers le répertoire parent
`md` ou `mkdir`	:	crée un nouveau répertoire
`rmdir`	:	supprime un répertoire, éventuellement avec l'ensemble de ses sous-répertoires si une option est utilisée, à utiliser avec précautions
`copy`	:	copie de fichier
`ren`	:	renomme le fichier spécifié
`del`	:	supprime le fichier

Ces commandes vous seront utiles sous DOSBox pour vous déplacer dans l'arborescence de vos fichiers. La liste complète des commandes DOS supportées par DOSBox est accessible ici : `https://www.dosbox.com/wiki/Commands`.

▶ Réalisez les manipulations suivantes en ligne de commande sous DOSBox :

— Placez-vous dans votre répertoire personnel (`Y:`)

— Listez son contenu

— Créez un répertoire « AO »

— Placez-vous dans ce répertoire et créez un sous-dossier « TP1 »

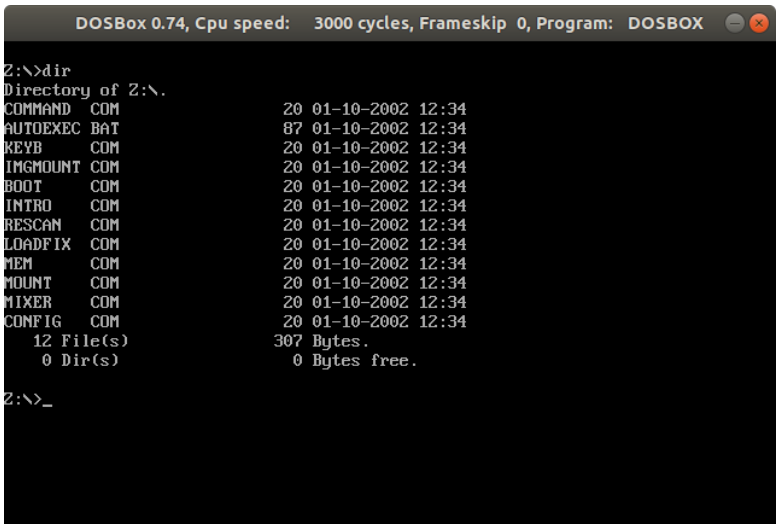


FIGURE 2 – Exemple d'utilisation de la commande `dir`

L'assembleur TASM

Pour pouvoir être exécuté par le microprocesseur, un programme assembleur doit être traduit en langage machine. Cette phase est appelée « assemblage » et est réalisée par un logiciel appelé « assembleur ». L'assembleur que nous allons utiliser est TASM (Turbo Assembler). Il en existe d'autres mais TASM est celui qui respecte le plus la syntaxe vue en cours. En suivant les instructions de la section précédente, vous avez déjà téléchargé et configuré TASM pour pouvoir l'utiliser sous DOSBox. Notez que les exécutables produits par TASM ne peuvent s'exécuter que sous l'émulateur DOSBox, les programmes étant uniquement dédiés à l'architecture du 8086.

La phase d'assemblage d'un programme produit un fichier « objet » contenant le code en langage machine. Ce fichier porte une extension `.obj`. Une phase d'édition des liens permet ensuite de créer un fichier exécutable par le système à partir d'un ou de plusieurs fichiers objet. Cet exécutable porte l'extension `.exe`. TASM fournit plusieurs outils pour réaliser ces étapes, ainsi que pour déboguer les programmes. Ces outils s'appellent respectivement : TASM (Turbo Assembler), TLINK (Turbo Linker) et TD (Turbo Debugger), et s'utilisent de la manière suivante :

— Pour assembler : `tasm nom_fichier_source.asm`

— Pour lier : `tlink nom_fichier_objet1.obj [nom_fichier_objet2.obj ...]`

— Pour déboguer : `td nom_programme.exe`

Exercice 1 – Premier programme en assembleur

Ce premier exercice vise à écrire un programme assembleur très simple et à en analyser la structure et la traduction en langage machine.

1. À partir de l'explorateur de fichiers d'Ubuntu, dans le dossier « TP1 », créez un fichier `EX1.ASM` et recopiez dans ce fichier le programme assembleur suivant :

```
1  assume  CS:code      ; on precise ici a l'assembleur que les
2                       ; instructions du programme se trouvent dans
3                       ; le segment appele code.
4
5  code    segment      ; definition d'une zone memoire de type "segment"
6
7  debut:               ; debut des instructions
8                       ; sequence d'instructions
9
10 code    ends         ; fin du segment de memoire dedie au code
11         end debut    ; fin du programme
```

2. À la ligne 8, avant le symbole ';', rajoutez l'instruction suivante :

```
mov AX, 1
```

L'instruction `mov` permet d'affecter une valeur à un registre ou à une variable (ici, on affecte la valeur décimale 1 au registre AX). Tout texte qui suit le symbole ';' représente

un commentaire, c'est-à-dire des informations qui ne seront pas traduites en langage machine.

À quoi servent ces commentaires ?

3. Vous allez maintenant remplacer l'instruction `mov AX, 1` par plusieurs instructions permettant de faire l'addition de $27h$ avec $21h$ dans le registre AL et de terminer le programme. Pour cela, vous allez réaliser 4 étapes :

   — affecter la valeur $27h$ au registre AL

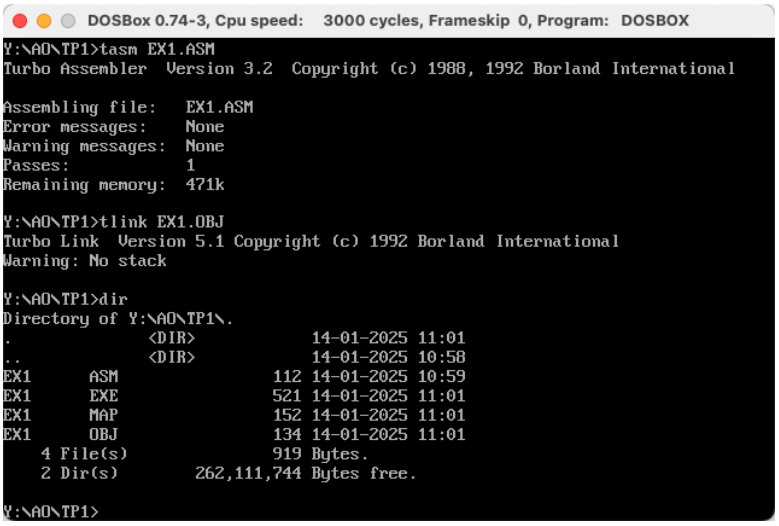
   — faire l'addition de AL et $21h$ avec l'instruction `add`

   — affecter la valeur $4C00h$ au registre AX

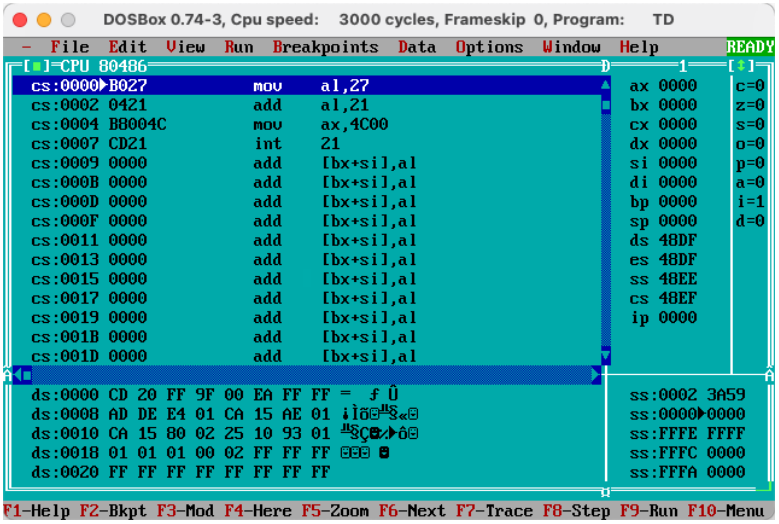
   — déclencher une interruption $21h$ avec l'instruction `int 21h`.

   Ces deux dernières instructions permettent de terminer « proprement » un programme et sont indispensables pour éviter le blocage du système.

4. Dans la fenêtre DOSBox, placez-vous dans le dossier « TP1 » ; compilez et liez votre programme. Vérifiez le contenu de votre répertoire « TP1 » :



5. Déboguez votre programme avec Turbo Debugger (commande `td EX1`) :



— Exécutez le programme en mode pas à pas et suivez la modification des différents registres (registres généraux et registres spécifiques, tels que le registre IP ou encore le registre d'état).

— En analysant la fenêtre correspondant au segment de code, retrouvez les instructions de votre programme. Comment ont-elles été codées en hexadécimal ? En déduire le rôle du logiciel TASM.

— Quel est le résultat de l'addition ?

Exercice 2 – Une addition de nombres en écriture décimale

1. Sur le même schéma que dans l'exercice précédent, créez un programme `EX2.ASM` réalisant l'addition des nombres 142 et 151 (en écriture décimale) à l'aide du registre `AX`.
2. Compilez, liez et déboguez votre programme.
3. Représentez le segment de code dans un tableau.
4. Vérifiez le résultat de l'addition.

Exercice 3 – La même addition ?

1. Sur le même schéma que dans l'exercice précédent, créez un programme `EX3.ASM` réalisant l'addition des nombres 142 et 151 (en écriture décimale) cette fois à l'aide du registre `AL`.
2. Compilez, liez et déboguez votre programme.
3. Représentez le segment de code dans un tableau.

4. Vérifiez le résultat de l'addition : quel registre permet de contrôler la correction du résultat ?

Exercice 4 – Une addition de nombres négatifs

1. Sur le même schéma que dans l'exercice précédent, créez un programme `EX4.ASM` réalisant l'addition des nombres -114 et -105 (en écriture décimale) à l'aide du registre `AL`.
2. Compilez, liez et déboguez votre programme.
3. Représentez le segment de code dans un tableau.
4. Vérifiez le résultat de l'addition : quel registre permet de contrôler la correction du résultat ?

Exercice 5 – La même addition de nombres négatifs ?

1. Sur le même schéma que dans l'exercice précédent, créez un programme `EX5.ASM` réalisant l'addition des nombres -114 et -105 (en écriture décimale) cette fois à l'aide du registre `AX`.
2. Compilez, liez et déboguez votre programme.
3. Représentez le segment de code dans un tableau.
4. Vérifiez le résultat de l'addition : quel registre permet de contrôler la correction du résultat ?