

TD 1

Prise en main de l'environnement DOS

Vous disposez de 4h¹ de laboratoire pour ce premier TD.

Ce TD a comme premier objectif de vous faire découvrir l'environnement de travail MS DOS. Les exercices de ce TD peuvent être fait dans un environnement DOS², dans un environnement MS Windows via son interpréteur de commandes ou dans un environnement Linux via un émulateur du genre dosemu³ ... à votre meilleure convenance.

Le second objectif de ce TD est la mise en oeuvre du premier jeu d'instructions vu au cours.

La lecture des parties "Assembleur DOS" et "Langage assembleur I" des transparents du cours est un prérequis au présent TD.

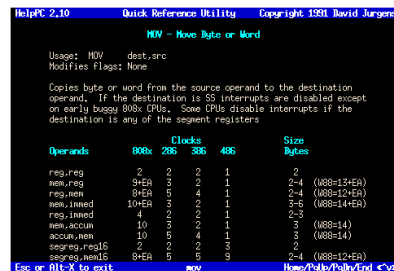
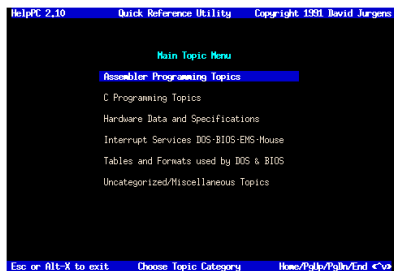
Préalables

Documentation

Presque, pour ne pas dire tous, les outils, logiciels, documents, ... dont vous aurez besoin se trouvent sur le site <http://esi.namok.be>⁴ qui dispose d'un miroir local à l'école dans le partage <\\srv-etd\edistri\pbt\www>.

Au laboratoire, vous aurez besoin de **helppc**, c'est un logiciel qui recense toutes les instructions assembleur ainsi que les interruptions⁵. Vous le trouverez sous forme d'une archive auto-extractible (**helppc.exe**), qu'il vous suffira d'exécuter dans le répertoire de votre choix. L'exécution créera un répertoire contenant le logiciel de documentation. Pratiquement, il vous suffira d'entrer les commandes⁶.

```
helppc[.exe]
cd helppc
helppc[.com]
```



1 Comme d'habitude, 4h au laboratoire et 4h à la maison.

2 Nous imaginons que personne parmi vous ne possède un tel environnement mais vous pouvez envisager d'installer MS-DOS 6.22 ou freedos dans une machine virtuelle.

3 Il existe également d'autres émulateurs (dosbox, ...) sous Linux.

4 Ce site préfère Firefox à MS IExplorer.

5 Nous verrons ça plus tard dans le cours.

6 La première commande ne devra plus être entrée par la suite puisque l'archive sera extraite.

Évaluation

Au terme du laboratoire vous serez évalués. Utilisez les cadres vides afin de noter vos réponses et remarques. Préparez également un code source assembleur commenté que vous construirez au fur et à mesure de ce TD et que vous présenterez lors de l'évaluation.

Édition de texte, compilation, édition de liens et exécution

Édition du texte

Pour éditer vos codes sources, vous aurez besoin d'un **éditeur de texte**. Nous déconseillons les éditeurs de texte qui n'ont pas vocation informatique tels que *Notepad* et consors. Nous déconseillons également l'éditeur de texte fourni avec MS-DOS, soit *edit* car il devient obsolète.

Nous conseillons un éditeur de texte fait pour la programmation et comme le meilleur moyen de connaître en profondeur un logiciel est de l'utiliser souvent, nous vous conseillons d'utiliser *vim*, disponible quelle que soit la plateforme que vous utilisez. Vous pouvez vous essayer à d'autres éditeurs de texte, il en existe beaucoup, l'important sera de faire un choix (un jour) afin d'en connaître **un** convenablement.

Pour installer *gVim* chez vous,

- sous *Linux*, installez le paquet **gvim** avec votre gestionnaire de paquets,
- sous *MS Windows*, rendez-vous sur le site de *vim* (<http://vim.org>) et téléchargez le fichier `gvim71.exe`, il se trouve également sur [mon site](#).

À l'école, vous travaillerez dans votre *drive Z*: dans lequel vous créerez un répertoire pour les labs, dans la suite nous dirons `Z:\dos`⁷. Vous pouvez déjà éditer votre premier programme assembleur, qui n'écrira même pas "Hello world", le sauver dans votre répertoire sous le nom `un.asm`. En assembleur sous MS-DOS, soyez prudent et n'utilisez pas de nom de fichier de plus de 8 caractères. Le problème de la limitation "8.3" est normalement résolu depuis MS-DOS > 6.22, mais nous sommes un peu paranos.

```
; =====
; un.asm
;
; Programme assembleur ne contenant que la structure du
; programme, et une manipulation de registre.
;
; Auteur : inconnu
; Date : janvier 2008
; =====

.MODEL small
.STACK 100h

.CODE
    mov AX,42

; ---- Epilogue ----
    mov AX,4C00h
    int 21h
END
```

Les deux dernières instructions permettent de terminer convenablement le programme et de rendre la main au système, elles seront vues plus en détails dans la suite du cours.

7 Vous l'organisez évidemment comme vous voulez,

Compilation

L'environnement de travail en assembleur se résume à quelques programmes, le premier étant le **compilateur**. Il existe divers compilateurs assembleur, nous utiliserons **tasm**.

Si le compilateur n'est pas installé, il faudra le faire ! Par convention, nous installerons "assembleur" dans le répertoire `C:\tasm` que vous pouvez créer pour l'occasion. Si le répertoire existe déjà vous pouvez l'effacer et refaire la manipulation (il est inutile de le faire à chaque labo) qui consiste en

- décompresser l'archive **tasm_td.zip**⁸ dans le répertoire `C:\tasm`,
- positionner correctement la variable d'environnement `PATH` afin que les exécutables contenus dans le répertoire `C:\tasm` puissent être trouvés. Pour ce faire ajouter au contenu actuel de la variable d'environnement `PATH` la valeur `C:\tasm`. Vous pourrez vérifier dans une console MS-DOS, que l'on obtient en lançant l'exécutable `command.com`, la valeur de la variable d'environnement `PATH` en entrant la commande `path`.

Placez-vous maintenant dans votre répertoire de travail ([Z:\dos](#)), un fichier `un.asm` s'y trouve, nous allons le compiler et faire apparaître *-magic-* un fichier `un.obj`. Pour rappel, les informations entre [] sont facultatives.

```
tasm un[.asm] /L/C/ZI
```

Les options permettront de déboguer le programme, elles signifient

- `/L` produit un listing du code source,
- `/C` produit une table des références croisées utilisée par le débogueur,
- `/ZI` produit une table des symboles également utilisée par le débogueur.

Édition de liens et exécution

L'éditeur de lien s'appelle **tlink**. Cette étape n'appelle pas de commentaires à ce stade. Il vous suffit d'entrer la commande afin de produire l'exécutable `un.exe`.

```
tlink un[.obj] /v
```

À nouveau, l'option `/v` est destinée à produire des informations utilisées par le débogueur.

Dès lors que l'exécutable est créé, il reste à le lancer, la commande suivante devrait faire l'affaire.

```
un[.exe]
```

Vous ne voyez rien, c'est normal, votre programme ne fait rien de visible⁹ mais nous y reviendrons.

Exécution via le débogueur

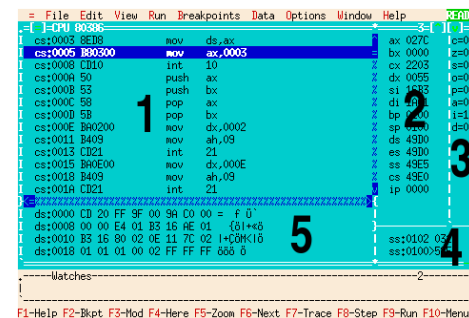
Le dernier programme que nous utiliserons dans le cadre de l'*assembleur sous MS-DOS*, est le **débogueur** fourni avec `tasm`. Un débogueur est un logiciel qui aide le programmeur à détecter les erreurs, les bogues, dans un programme ou simplement qui permet de tester un programme. Le débogueur offre la possibilité d'exécuter un programme *pas-à-pas*, de placer des *points d'arrêt* dans un programme, de visualiser le contenu des variables (et des registres dans notre cas) voire de les modifier, de se déplacer dans un programme, etc.

Le débogueur fourni avec `tasm` s'appelle **turbo debugger**. Il possède une interface graphique¹⁰ permettant de visualiser "le CPU".

⁸ Que vous trouverez sur le site du cours, c'est déjà dit.

⁹ Il ne fait peut-être rien, mais il le fait bien, du moins si vous êtes arrivés là sans erreur.

¹⁰ Vous verrez qu'elle n'est pas toute récente et qu'il faut parfois se méfier du fonctionnement de la souris mais elle permet de bien comprendre le fonctionnement du processeur.



Pour obtenir cette vue, entrer `[Alt-V]` et choisissez CPU, on y distingue

1. le code source; les instructions en hexadécimal, suivies de leur signification en assembleur,
2. les registres et leur valeur,
3. les flags et leur valeur,
4. le sommet de la pile (voir plus tard),
5. la mémoire (voir plus tard)

Notez bien les touches de fonction (`F7` permet d'exécuter le programme instruction par instruction). Vous pourrez également utiliser les touches `[Ctrl]` et `[Alt]` afin de découvrir d'autres menus.

Pour tracer votre programme, et donc lancer turbo debugger, il vous suffit d'entrer la commande

```
td un[.exe]
```

Instructions élémentaires

Instruction mov

Reprenez le programme `un.asm` et ajoutez-y les instructions suivantes après l'instruction `mov AX, 42`

```
mov BL, 17
mov CX, 17h
mov DH, 'A'
mov DX, BX
```

Pour rappel, certains registres (16 bits) peut se découper en deux registres (8 bits), par exemple le registre `BX` peut être vu comme contenant les registre `BL` et `BH`, respectivement comprenant les 8 bits de poids faibles de `BX` et les 8 bits de poids forts.

Ajoutez la directive `. 386` dans l'entête de votre programme et l'instruction

```
mov EAX, 12345678h
```

Maintenant, posez-vous les questions et observez,

- Comment évolue le registre `IP` ? S'incrémente-t-il toujours de 1 ?
- Pour la première instruction, `mov AX, 42`, où voyez-vous que le registre est modifié ?
- Pourquoi après avoir exécuté la deuxième instruction, je ne vois pas 17 dans le registre `BL` mais après la troisième instruction j'ai bien 17 dans le registre `CX` ?
- **Après** avoir exécuté la seconde instruction, modifiez la valeur de `BL` (par 18) au sein de `td`¹¹ (pour ce faire, observez les différents menus).

¹¹ `td` est le petit nom de Turbo Debugger, vous l'aurez compris.

- Que vaut CH après exécution de la troisième instruction ?
- Quid de la valeur des *flags* ? Pourquoi ?
- ...

Instructions add et sub

Écrivez quelques instructions permettant d'additionner 3 et 14. Vous ferez cela de trois manières différentes:

- en utilisant les registres AX et BX,
- en utilisant les registres AL et BL,
- en utilisant les registres AL et AH.

Faites la même chose pour additionner -3 et 14. Observez les résultats et les différences entre les trois manières de faire dans td.

Faites la même chose pour additionner 314 et 15. Observez les résultats et les différences entre les trois manières de faire dans td. *Quid* de la valeur des *flags* ?

Écrivez quelques instructions permettant de soustraire 11111111b à 00000010b, interprétez le résultat.

Choisissez deux opérandes permettant de positionner le *flag zero* après une soustraction. Faites de même pour positionner le *flag signe* et ensuite le *flag overflow*.

Instructions mul et imul

Les instructions suivantes permettent de multiplier 3 par 14.

```
mov AL, 3
mov BL, 14
mul BL
```

Ajoutez ces instructions à votre programme et observez les résultats dans td.

- Où se trouve la valeur 42 ?
- Que se passe-t-il si l'on remplace BL par BX dans la seconde instruction ?
- Que se passe-t-il si j'ajoute avant la première instruction, l'instruction `mov AH, 15h` ?

Si vous remplacez 3 par -3 dans la première instruction, où se trouve le résultat de la multiplication et que vaut-il ? Si ce n'est pas -42, qu'obtenez-vous comme réponse, comment obtenir la bonne réponse et où et comment la lire ?

Écrivez quelques instructions permettant de multiplier 314 par 15.

Est-il possible de multiplier AAAAh par BBBBh ? ¹² Quid de la valeur des *flags* ? Pourquoi ?

¹² Ma calculatrice peut le faire, elle obtient 7D26D82Eh

Instructions div et idiv

Pour rappel, l'instruction `div` en assembleur n'accepte qu'un seul opérande. Comme chacun sait pour faire une division, il faut un *dividende* et un *diviseur*. Le *dividende* n'est pas renseigné dans l'instruction, seul le *diviseur* l'est. Et une division fournit un *quotient* et un *reste*.

```
mov AX, 7
mov BL, 2
div BL
```

Quelle division réalisent ces instructions ? Visualisez dans td où se trouvent le dividende, le diviseur, le quotient et le reste.

Testez votre programme en remplaçant 7 et 2 par 7 et -2, -7 et 2 et par -7 et -2. Faites de même en remplaçant `div` par `idiv`. Observez bien les résultats.

Écrivez les instructions permettant d'effectuer la division de 12345h par 2h, observez vos résultats avec td.

Instructions cbw et cwd

Écrivez les instructions suivantes, observez les effets et interprétez les résultats.

```
mov AX, 0
mov AL, 42
cbw
mov AX, 0
mov AL, -42
cbw
mov AX, 0
mov AL, 255
cbw
```

- Dans chaque cas, précisez ce que vaut AX et AL après l'instruction `cbw`.
- Quelles différences entre

```
mov AL, 42
cbw
```

et

```
mov AH, 0
mov AL, 42
```

- Même questions dans les autres cas.

Notions de variable

En assembleur, il est possible de définir des variables. Ajoutez une section de données à votre code (elle se place avant la directive `.CODE`).

```
.DATA  
b1 DB 3  
b2 DB ?  
w1 DW 14  
w2 DW 000Fh
```

- Essayez d'ajouter l'instruction `mov AX, b1` à votre code et observez où se trouve la valeur 3. Corrigez les erreurs éventuelles afin de visualiser cette valeur 3.
- Écrivez l'(es) instruction(s) permettant de copier le contenu de `b1` dans `b2`.
- Visualisez dans la partie mémoire de td (la partie 5 sur la capture d'écran (*screenshot*)) où se trouvent les variables, repérez les valeurs; 3 (rien) 14 et Fh. Pour ce faire faites un *goto* à l'adresse `DS:0000h` et n'oubliez pas que tout est écrit en hexadécimal.
 - dans cette vue, si je dis "**petit boutiste**" ai-je éternué ? Si non, expliquez.
 - traçez votre programme et observez le changement de valeur dans cette vue lorsque vous modifiez la valeur de `b2`.
- Écrivez les instructions permettant d'ajouter le contenu de `b2` à `w1`.
- Écrivez les instructions permettant de multiplier `w1` par 3.
- Dans ce contexte, quel est le résultat de l'opération

```
mov DS: [2], 42
```