

## MI01 TP4 - Prise en main de l'environnement de développement assembleur et premiers programmes

Le but de ce TP est de prendre en main l'environnement de développement des TP ainsi que les outils de débogage, et de commencer à écrire quelques programmes.

### 1 Prise en main de l'environnement de développement

#### 1.1 L'environnement de développement

L'outil utilisé dans le cadre des TP assembleur est l'environnement de développement intégré Microsoft Visual Studio 2005, qui fournit un compilateur C/C++, l'assembleur MASM 8.0 (Microsoft Macro Assembler), un éditeur de liens et une interface graphique qui permet l'édition des fichiers sources, qui automatise la construction de programmes et qui inclut un débogueur afin d'en suivre l'exécution pas-à-pas, si nécessaire.

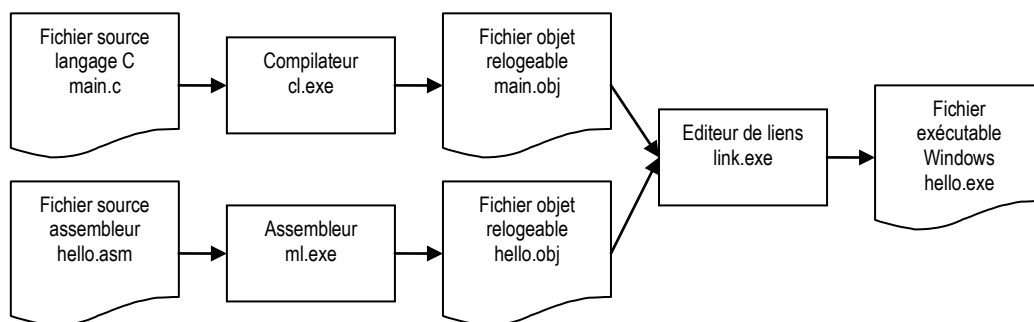


Figure 1 - Chaîne de construction du programme exécutable

Une application se compose de plusieurs modules, à chaque module étant associé un fichier source C ou assembleur. L'environnement de développement est destiné à transformer ces différents modules en une application unique exécutable par le processeur. Les différentes étapes de la construction d'un fichier exécutable (Figure 1) sont les suivantes :

- Compilation par *cl.exe* des modules en langage C
- Assemblage par *masm.exe* des modules en langage d'assemblage
- Regroupement de tous les modules par l'éditeur de liens *link.exe* pour former un seul fichier exécutable.

Dans le cadre des TP de MI01, le module en langage C vous sera fourni si nécessaire. Dans ce TP, il n'y a pas de module C.

#### Mise en place

- 1) Sur le site Moodle de l'UV, téléchargez le fichier projet du TP « IA32TP1.zip » et copiez son contenu dans votre dossier de projet (sur le disque Z:)
- 2) Lancez l'environnement de développement intégré *Microsoft Visual Studio 2005*.
- 3) Dans le menu « Fichier », choisissez « Ouvrir→Projet/Solution... », naviguez jusqu'au dossier où vous avez décompressé le fichier et sélectionnez le fichier *tp\_asm.sln*. Votre projet est prêt à être utilisé.

Il faut maintenant ajouter un fichier source.

- 1) Sélectionnez le menu « Projet→Ajouter un élément existant... ».
- 2) Dans la fenêtre qui s'ouvre, choisissez *programme.asm*.

Ouvrez le fichier en double-cliquant sur son nom dans l'explorateur de solutions.

## 1.2 Structure d'un fichier source MASM

Un fichier source MASM se compose de plusieurs sections, qui vont permettre de définir le modèle de segmentation mémoire utilisé par le programme, ainsi que le contenu du segment de données (variables) et le contenu du segment de code (instructions). Il comporte de plus des directives d'assemblage à destination de l'assembleur qui modifient son comportement.

Une référence complète est disponible sur le site MSDN de Microsoft à l'adresse [http://msdn.microsoft.com/en-us/library/afzk3475\(v=VS.80\).aspx](http://msdn.microsoft.com/en-us/library/afzk3475(v=VS.80).aspx).

L'exemple suivant détaille la structure d'un fichier source.

### • Entête

```
; programme.asm
;
; MI01 - TP Assembleur 1
;
; Affiche un caractère à l'écran
TITLE programme.asm

.686
.MODEL FLAT, C

EXTRN      putchar:NEAR
EXTRN      getchar:NEAR
```

La directive **TITLE** permet de définir le nom du module. Les commentaires sont précédés d'un point-virgule, tous les caractères jusqu'à la fin de la ligne sont ignorés.

La directive **.686** autorise l'utilisation du jeu d'instruction complet des Pentium II et suivants.

La directive **.MODEL** spécifie le modèle de segmentation utilisé. Dans le cas d'un programme Windows, on utilise le modèle **FLAT**. **C** indique que la convention d'appel de fonctions est celle du langage C (passage de paramètres par la pile, retour dans EAX, l'appelant est chargé de dépiler les paramètres).

Suivent les déclarations de fonctions externes, qui sont fournies par d'autres modules. Ici on informe l'assembleur de l'existence d'une fonction externe (**EXTRN *putchar***) fournie par la bibliothèque C et que cette fonction se situe dans le même segment de code que le reste du programme (**NEAR**). On peut ajouter autant de déclarations EXTRN qu'on veut.

### • Segment de données

```
.DATA
cara          DB      'A'
```

Le début du segment de données est indiqué par la directive **.DATA**. C'est cette partie qui contient la définition des variables. Le format général de ces définitions est le suivant :

*Etiquette*      *taille*    *valeur[,valeur,[valeur,...]]*  
*Etiquette*      *taille*    *n DUP(valeur)*

La forme avec DUP permet de répéter la définition *n* fois.

La taille est spécifiée comme suit :

DB (Data Byte)	8 bits
DW (Data Word)	16 bits
DD (Data Double Word)	32 bits
DQ (Data Quad Word)	64 bits

### Exemples

```
cara          DB      'A'      ; Réserve 1 octet, initialisé à A
chaine        DB      "toto"   ; Réserve 4 octets, initialisés à t o t o
```

nombre	DW	0FFAAh ; Réserve 1 mot, initialisé à (FFAA) <sub>16</sub>
tableau	DW	0, 1, 2 ; Réserve 3 mots, initialisés à 0, 1 et 2
inconnu	DD	? ; Réserve 1 double mot, non initialisé
tab2	DB	6 DUP(?) ; Réserve 6 octets non initialisés
tab3	DQ	8 DUP(0) ; Réserve 8 quadruple mots, initialisés à 0

Les formats des constantes sont les suivants :

'A'	Caractère
"AA"	Chaîne de caractères
1010b	Entier en base 2
755o	Entier en base 8
12	Entier en base 10
-12	Entier en base 10, complémenté à 2.
1234h	Entier en base 16
-0feh	Entier en base 16, complémenté à 2.

- Segment de code

```
.CODE
PUBLIC      main
main        PROC

    ; Conversion du caractère en un double mot
    movzx   eax, byte ptr[car]

    ; Préparation de l'appel à la fonction de
    ; bibliothèque 'C' putchar(int c) pour afficher
    ; un caractère. La taille du type C 'int' est
    ; de 32 bits sur IA-32. Le caractère doit être fourni
    ; sur la pile.

    push    eax            ; Caractère à afficher
    call    putchar        ; Appel
    add     esp, 4         ; Nettoyage de la pile après appel
    ; Fin de l'appel à putchar

    call    getchar        ; Attendre l'appui sur « Entrée »

    ret                                ; Retour au module C

main        ENDP
```

Le début du segment de code est indiqué par la directive **.CODE**. C'est cette partie qui contient les instructions du programme exécutées par le processeur. Dans notre cas, le point d'entrée du programme (la première instruction exécutée) est fournie par la bibliothèque C, qui à son tour fait appel à la fonction *main*. La définition d'une fonction se fait comme suit :

```
etiquette    PROC
...
etiquette    ENDP
```

L'étiquette, qui représente l'adresse de la procédure, doit être accessible par le module en langage C pour qu'il puisse y faire appel. La directive **PUBLIC** permet de rendre une étiquette connue de tous les modules.

Le format d'une ligne de programme est le suivant :

*etiquette: mnémonique opérandes ; Commentaire*

Il est parfois nécessaire de spécifier la taille d'un opérande mémoire explicitement, quand l'assembleur n'est pas capable de la déterminer :

```
inc    byte ptr[donnee]    ; Donnée mémoire 8 bits
inc    word ptr[donnee]    ; Donnée mémoire 16 bits
inc    dword ptr[donnee]   ; Donnée mémoire 32 bits
```

Il est possible de charger directement l'adresse d'une variable dans un registre 32 bits :

```
mov    eax, offset nombre    ; eax <= adresse mémoire de « nombre ».
```

### • Fin du module

END

## 1.3 Assemblage et exécution du programme

Dans le menu « Générer », lancez la création de l'image exécutable par la commande « Générer la solution ». Si tout se passe bien, exécutez le programme en sélectionnant « Déboguer→Exécuter sans débogage ».

## 1.4 Utilisation du débogueur

Le débogueur permet de suivre le déroulement du programme pas à pas en affichant l'état complet du processeur à chaque pas (contenu des registres, drapeaux...).






Afin de pouvoir l'utiliser, il faut que le programme s'interrompe lors de l'exécution. On utilise des points d'arrêt à cette fin. Lorsque le processeur rencontre un point d'arrêt lors de l'exécution des instructions, il bloque l'exécution du programme en cours et transfère le contrôle à un programme superviseur, dans ce cas le débogueur.

Pour placer/supprimer un point d'arrêt, il suffit de cliquer dans la zone grise à gauche de l'éditeur, en face d'une instruction. Un point rouge (●) apparaît, indiquant que le programme stoppera à cette instruction (ou à la première ligne contenant une instruction dans l'éditeur après ce point).

Dans le programme d'exemple, placez un point d'arrêt au niveau de l'instruction « MOVZX », puis lancez l'exécution (commande ▶) après avoir vérifié que « Debug » est sélectionné dans la liste déroulante à droite de la commande.

Une fois le programme stoppé, un curseur (⏏) indique l'instruction qui sera exécutée ensuite.

L'exécution peut alors être poursuivie au moyen des commandes suivantes :

	Poursuit l'exécution normale du programme à pleine vitesse
	Exécute une seule instruction (pas à pas)
	Exécute une seule instruction mais exécute les sous-programmes à pleine vitesse
	Met fin à l'exécution du programme
	Relance le programme au début


Affichez l'état des registres (menu « Déboguer→Fenêtres→Registres). Dans la fenêtre des registres, un clic-droit permet d'obtenir un menu de sélection des informations affichées : cochez « CPU », « Segments d'UC » et « Indicateurs ».

Exécutez le programme instruction par instruction (⏏ ou F10). A chaque instruction, le débogueur met à jour l'état du processeur. Vous pouvez définir d'autres points d'arrêt, mais vous ne pouvez pas modifier la valeur des registres.

## 2 Affichage de chaînes de caractères

### 2.1 Programme « Hello World »

Dans l'explorateur de solutions, sélectionnez *programme.asm* puis enlevez-le du projet (menu « Projet→Exclure du projet »), puis ajoutez *hello1.asm* au projet (menu « Projet→Ajouter un élément existant... »).

Vous pouvez afficher l'ensemble des fichiers exclus en activant « Afficher tous les fichiers » dans l'explorateur de solutions (bouton ).

**Q1** - Dans le fichier *hello1.asm*, définissez une nouvelle variable *msg* de type chaîne de caractères qui contient "bonjour tout le monde", ainsi qu'une variable *longueur* qui contient la longueur de la chaîne. Quelle doit être la taille en bits de *longueur*, sachant qu'on va la comparer au registre *ebx* ?

Le sous-programme *main* est maintenant le suivant :

```
main      PROC

    push    ebx                ; Sauvegarde pour le code C

    mov     ebx, 0

    ; On suppose que la longueur de la chaîne est non nulle
    ; => pas de test de la condition d'arrêt au départ.
suivant:  movzx  eax, byte ptr[ebx + msg]

    ; Préparation de l'appel à la fonction de
    ; bibliothèque 'C' putchar(int c) pour afficher
    ; un caractère. La taille du type C 'int' est
    ; de 32 bits sur IA-32. Le caractère doit être fourni
    ; sur la pile
    push    eax                ; Paramètre de putchar
    call    putchar            ; Appel
    add     esp, 4              ; Nettoyage des paramètres
    ; Fin de l'appel à putchar(int c).

    inc     ebx                ; Caractère suivant
    cmp     ebx, [longueur]; Toute la longueur ?
    jne     suivant            ; si non, passer au suivant

    call    getchar            ; Attente d'appui sur une touche
    pop     ebx                ; Restauration pour le code C

    ret                                ; Retour au programme C.

main      ENDP
```

**Q2 - Décrivez le fonctionnement de ce programme. Quel est l'algorithme utilisé ? Que représente ebx ?**

Note : les fonctions de bibliothèque C ne préservent que les registres esi, edi et ebx. Il est donc nécessaire de sauvegarder les autres registres utilisés par le programme lors de l'appel à *putchar*.

Observez le déroulement du programme au moyen du débogueur.

## 2.2 Optimisation du programme

Copiez le programme *hello1.asm* dans un nouveau fichier que vous appellerez *hello1op.asm*. Modifiez le projet en conséquence en ajoutant *hello1op.asm* et en excluant *hello1.asm*

**Q3 - On souhaite maintenant optimiser ce programme. En utilisant un registre supplémentaire pour stocker l'adresse de la fin du message, comment éliminer la ligne « cmp ebx, [longueur] » ? Expliquer le fonctionnement de votre programme. Pour quelle(s) raison(s) chercher à éliminer cette ligne de programme ?**

Indications : dans les adressages indirects indexés, l'index peut être négatif. Ainsi, on suppose :

```
movzx eax, byte ptr[esi + ebx].
```

Si esi contient 21, et que ebx contient -20, l'adresse effective de l'opérande mémoire sera 1.

Pour charger l'adresse de *msg* dans un registre, utilisez l'instruction *lea*.

## 2.3 Chaîne de taille variable

Copiez le programme *hello1.asm* dans un nouveau fichier *hello2.asm*. Modifiez le projet en conséquence.

On veut maintenant se passer de la longueur de la chaîne à afficher. A cet effet, on marque la fin de la chaîne de caractères par un caractère nul (0). Modifiez la valeur initiale de la chaîne *msg* comme suit, et supprimez la variable *longueur* :

```
msg      db      "bonjour tout le monde", 0
```

**Q5 - Quel est l'algorithme à réaliser ? Modifiez le programme *hello2.asm* pour implémenter cet algorithme.**

### 3 Conversion et affichage de nombres

Ajoutez au projet le fichier source *conversion.asm*. N'oubliez pas d'exclure *hello2.asm*. L'objectif est maintenant d'écrire un programme qui réalise l'affichage d'un nombre à l'écran.

#### 3.1 Conversion d'un nombre non signé en base 10

Afin de convertir le nombre en une chaîne de caractères affichables, on utilise le principe des divisions successives par la base pour le décomposer en une suite de chiffres :

$$\begin{array}{rcl}
 257 & | & 10 \\
 7 & \leftarrow & 7 \quad | \quad 25 \quad | \quad 10 \\
 5 & \leftarrow & \quad 5 \quad | \quad 2 \quad | \quad 10 \\
 2 & \leftarrow & \quad \quad 2 \quad | \quad 0
 \end{array}$$

**Q1** - On stocke les chiffres convertis dans une chaîne de *n* caractères, *n* à définir. Chaque caractère occupe exactement un octet en mémoire. Sachant que *nombre* est exprimé sur 32 bits, donner l'expression de la valeur exacte de *n*. Quelle valeur entière doit-on utiliser dans le programme ? Remplacez les deux « x », ligne 18 dans le programme, par la valeur que vous avez calculée.

**Q2** - Proposez un programme en assembleur IA-32 qui réalise la conversion de *nombre* et stocke **chaque caractère successif obtenu par le reste de la division en commençant à l'adresse chaîne**. Vous prendrez garde à la condition d'arrêt des itérations ainsi qu'à la cohérence de taille des opérandes, en particulier lors de la division et de la conversion du reste en chiffre affichable : *nombre* est stocké sur 32 bits, chaque caractère de la chaîne doit faire un octet.

Vérifiez le bon fonctionnement de votre programme au moyen du débogueur. Vous pouvez afficher le contenu de la mémoire en sélectionnant le menu « Déboguer→Fenêtres→Mémoire→Mémoire 1 ». Dans le panneau « Mémoire 1 », entrez par exemple *&chaîne* dans la ligne « Adresse » pour afficher le contenu de la mémoire à partir de l'adresse de *chaîne*.

**Q3** - Quelle est la particularité de la chaîne que votre programme construit ?

#### 3.2 Affichage du nombre

**Q4** - En tenant compte de la particularité précédente, complétez de programme pour qu'il affiche à l'écran le nombre converti de manière lisible. Pour l'affichage proprement dit, utilisez le sous-programme *putchar* comme les exercices précédents (vous pouvez copier / coller le code nécessaire).

#### 3.3 Exercice supplémentaire : prise en compte du signe

**Q5** - On veut maintenant pouvoir traiter les nombres représentés en complément à 2. Modifiez votre programme pour qu'il prenne en compte le signe du nombre et qu'il préfixe l'affichage par le caractère '-' au cas où celui-ci est négatif. **Attention** : *putchar* modifie la valeur du registre *eax*.

#### 3.4 Exercice supplémentaire : base quelconque entre 2 et 36

On ajoute le tableau de caractères suivant dans les données du programme :

chiffres	db	"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
----------	----	--

**Q6** - Comment convertir le reste de chaque division entière du programme de conversion en caractère affichable au moyen de ce tableau ? Modifiez votre programme en conséquence.

**Q7** - Quelle est maintenant la taille maximale de la chaîne nécessaire pour pouvoir stocker tous les chiffres quelle que soit la base choisie ?

**Q8** - Donnez au moyen de votre programme l'expression de *nombre* en base 32.