Université de Technologie de Compiègne

MI01

Rapport de TP

4 - Prise en main de l'assembleur et premiers programmes

Automne 2014

Romain PELLERIN - Kyâne PICHOU
Groupe 1
7 décembre 2014

Table des matières

1 Introduction

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 en assembleur. On étudiera pour commencer la structure d'un fichier assembleur MASM, puis on travaillera sur l'affichage d'un "Hello World !". Enfin, on réalisera un programme plus complexe de conversion de nombres.

2 Exercices

2.1 Structure d'un fichier source MASM

Pour commencer on utilise le fichier programme.asm pour découvrir la structure d'un fichier source MASM. En entête, on découvre les directives **TITLE**, **.686** et **.MODEL**. **TITLE** donne un nom à notre programme assembleur, **.686** indique que l'on utilise le jeu d'instruction **.686** et **MODEL** spécifie notre modèle de segmentation. Il prend la valeur **FLAT** (pour Windows) et **C** (spécifie la convention d'appel de fonctions).

Ensuite, **EXTERN** et **.DATA** servent à déclarer les fonctions externes et les segments de données.

On compile et on exécute le code suivant :

Assembleur

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

TITLE programme.asm

.686
.MODEL FLAT, C

EXTERN     putchar:NEAR
EXTERN     getchar:NEAR

.DATA

cara DB 'A' ; On déclare une variable initialisée à 'A'


.CODE

; Sous-programme main, automatiquement appelé par le code de
; démarrage 'C'

PUBLIC     main
main       PROC

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

           ; 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 de putchar (fonction C qui va réaliser l'
               affichage d'un caractère)
            add     esp, 4      ; Nettoyage de la pile après appel
            ; Fin de l'appel à putchar

            call    getchar     ; Attente de l'appui sur "Entrée"

            ret                 ; Retour au code de démarrage 'C'

main        ENDP

            END
```

Le code ouvre un terminal dans lequel s'affiche tout simplement le caractère *cara*.

2.2 Affichage de chaînes de caractères

2.2.1 Programme "Hello World"

On utilise maintenant le fichier *hello1.asm* pour réaliser un "Hello World!". On définit une variable *msg* qui contiendra notre chaine de caractère et une variable *longueur* contenant la longueur de la chaine. Cette variable longueur sera comparée avec le registre *ebx* afin de savoir si l'on a parcouru toute la chaine de caractère ou non. *ebx* étant de taille 32 bits, longueur devra faire 32 bits. On a le code suivant :

Assembleur

```
; hello1.asm
;
; MI01 - TP Assembleur 1
;
; Affiche une chaîne de caractères à l'écran

TITLE hello1.asm

.686
.MODEL FLAT, C

EXTERN      putchar:NEAR
EXTERN      getchar:NEAR

.DATA

; Ajoutez les variables msg et longueur ici
msg         DB  "bonjour tout le monde"
longueur    DD  21

.CODE
```

```
; Sous-programme main, automatiquement appelé par le code de
; démarrage 'C'
PUBLIC     main
main       PROC

           push    ebx           ; Sauvegarde pour le code 'C'
           ; Met la valeur de ebx dans la pile

           mov     ebx, 0
           ; ebx <- 0 car ebx va servir de compteur

           ; 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]
           ; on met le premier caractère de notre string dans eax

           ; 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. Cf cours sur les sous-programmes.
           push    eax           ; Caractère à afficher
           ; Met la valeur de eax dans la pile

           call    putchar       ; Appel de putchar
           ; Affiche sur la console la valeur du sommet de la pile

           add     esp, 4        ; Nettoyage de la pile après appel
           ; On décale le pointeur de sommet de pile

           ; Fin de l'appel à putchar

           inc     ebx           ; Caractère suivant
           ; Incrémente ebx de 1

           cmp     ebx, [longueur] ; Toute la longueur ?
           ; Compare ebx (qui s'incrémente à chaque tour) à notre "variable"
              longueur qui vaut 21

           jne     suivant       ; si non, passer au suivant
           ; Si pas égaux on saute à l'étiquette 'suivant'

           call    getchar       ; Attente de l'appui sur "Entrée"
           pop     ebx
           ; Depile la pile et mets l'élément dans ebx

           ret                   ; Retour au code de démarrage 'C'

main       ENDP

           END
```

Ce programme doit afficher la chaine de caractère *msg*. Pour cela on initialise le registre *ebx* à 0 : **il nous servira de compteur** de parcours de la chaîne de caractère. Le programme s'articule autour d'une boucle. Tout d'abord on récupère dans *eax* le caractère de la chaîne à "l'indice" *ebx* puis on affiche le caractère (*push* puis *putchar*). Ensuite on incrémente *ebx* de 1. Si *ebx* et longueur sont égaux (et donc qu'on a affiché tout les caractères) alors on se rend à la fin du programme (avec attente de l'appui sur "Entrée"). Sinon on reboucle pour afficher le caractère suivant.

2.2.2 Optimisation du programme

On cherche maintenant à optimiser le programme en supprimant la ligne `cmp ebx, [longueur ]`. En effet cette instruction de comparaison est coûteuse et s'exécute à chaque tour de boucle.

On utilisera donc un registre supplémentaire *esi* qui contiendra l'adresse du dernier caractère de notre message, et le registre de compteur *ebx* sera initialisé à -[nbr de caractères]. Ainsi on affichera, à chaque tour de boucle, le caractère de *msg* d'indice `[ebx+esi]`. Ce procédé nous permet de ne plus avoir à faire de comparaison. En effet on incrémente *ebx* à chaque tour de boucle, donc après l'affichage du dernier caractère, le résultat de l'incrémentation sera 0. Cela lèvera donc un *flag* d'état automatiquement qui, détecté par notre programme, entrainera la fin du programme. Ceci est moins couteux en terme de cycles processeur.

Notre implémentation dans le code ci-dessous :

Assembleur

```
; hello1op.asm
;
; MI01 - TP Assembleur 1
;
; Affiche une chaîne de caractères à l'écran

TITLE hello1op.asm

.686
.MODEL FLAT, C

EXTERN     putchar:NEAR
EXTERN     getchar:NEAR

.DATA

; Ajoutez les variables msg et longueur ici
msg        DB  "Hello World!"
longueur   DD  12

.CODE

; Sous-programme main, automatiquement appelé par le code de
; démarrage 'C'
PUBLIC     main
main       PROC

           push   ebx            ; Sauvegarde pour le code 'C'
           ; Met la valeur de ebx dans la pile
```

```
            mov ebx, [longueur]
            lea esi, [msg+ebx]
            ; esi contient l'adresse du dernier caractère

            neg ebx
            ; ebx contient -12

            ; 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 + esi]
            ; on met le premier caractère de notre string dans eax

            ; 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. Cf cours sur les sous-programmes.
            push   eax           ; Caractère à afficher
            ; Met la valeur de eax dans la pile

            call   putchar       ; Appel de putchar
            ; Affiche sur la console la valeur du sommet de la pile

            add    esp, 4        ; Nettoyage de la pile après appel
            ; On décale le pointeur de sommet de pile

            ; Fin de l'appel à putchar

            inc    ebx           ; Caractère suivant
            ; Incrémente ebx de 1

            jnz    suivant       ; si non, passer au suivant
            ; Si le flag égal à 0 n'est pas levé on saute à l'étiquette 'suivant'

            call   getchar       ; Attente de l'appui sur "Entrée"
            pop    ebx
            ; Depile la pile et mets l'élément dans ebx

            ret                  ; Retour au code de démarrage 'C'

main        ENDP

            END
```

2.2.3 Chaîne de taille variable

On chercher maintenant à se passer de la variable *longueur* en détectant automatiquement la fin de la chaîne de caractère. Pour cela il faudra que notre chaîne comporte un caractère de fin nul comme ci-dessous :

Assembleur

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

Notre algorithme est très simple. Nous reprenons la totalité du code précédent en rajoutant uniquement une comparaison. Après avoir mis le caractère dans eax on vérifie que celui-ci ne soit pas égal à 0. Si c'est le caractère 0, dans ce cas le programme saute à la fin. On retire donc l'instruction *jnz suivant* du précédent programme pour utiliser *jmp suivant* et ainsi reboucler sans aucune condition.

Assembleur

```
; hello2.asm
;
; MI01 - TP Assembleur 1
;
; Affiche une chaîne de caractères à l'écran

TITLE hello2.asm

.686
.MODEL FLAT, C

EXTERN     putchar:NEAR
EXTERN     getchar:NEAR

.DATA

; Ajoutez les variables msg et longueur ici
msg        DB  "bonjour tout le monde", 0
; 0 sera en fin de chaine msg

.CODE

; Sous-programme main, automatiquement appelé par le code de
; démarrage 'C'
PUBLIC     main
main       PROC

           push    ebx            ; Sauvegarde pour le code 'C'
           ; Met la valeur de ebx dans la pile

           mov     ebx, 0
           ; ebx <- 0 car ebx va servir de compteur

           ; 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]
           ; on met le premier caractère de notre string dans eax

           ; 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. Cf cours sur les sous-programmes.
```

```
            cmp eax, 0

            jz  fin ; Si égal à 0, on saute

            push    eax             ; Caractère à afficher
            ; Met la valeur de eax dans la pile

            call    putchar         ; Appel de putchar
            ; Affiche sur la console la valeur du sommet de la pile

            add     esp, 4          ; Nettoyage de la pile après appel
            ; On décale le pointeur de sommet de pile

            ; Fin de l'appel à putchar

            inc     ebx
            jmp     suivant

fin:         call    getchar
            pop     ebx

            ret

main        ENDP

            END
```

2.3 Conversion et affichage de nombres

2.3.1 Conversion d'un nombre non signé en base 10

Nous devons réaliser ici un programme qui va afficher un nombre. Pour cela on récupèrera caractère après caractère la totalité des chiffres composants ce nombre. On utilisera la méthode de divisions sucessives afin de récupérer tout les chiffres. On divise chaque quotient par la base que l'on veut utiliser (ici 10). En mettant bout à bout chacun des restes des ces divions ont obtiens (à l'envers) le nombre que l'on a converti.
Sachant que *nombre* est sur 32 bits, alors la plus grande valeur possible sera 2^{32}. Ce (grand) nombre s'écrit en 10 chiffre, il faudra donc mettre *chaine* à 10.
On se propose de coder ceci de la manière ci-dessous :

Assembleur

```
TITLE conversion.asm
.686
.MODEL FLAT, C
EXTERN      putchar:NEAR
EXTERN      getchar:NEAR
.DATA
nombre      dd      257         ; Nombre à convertir
chaine      db      10 dup(?)
.CODE
; Sous-programme main, automatiquement appelé par le code de
```

```
; démarrage 'C'
PUBLIC     main
main       PROC
                push eax         ; sauvegarde des registres
                push ebx
                push ecx
                push edx

                mov eax,[nombre] ; nombre dans eax
                lea ebx,[chaine] ; adresse de chaine dans ebx
                mov ecx,10       ; On effectuera des divisions par 10
boucle:

                xor edx,edx      ; mise à 0 du registre
                div ecx          ; On effectue la divison : le reste sera dans
                   edx et le quotient dans eax

                mov [ebx],edx    ;On déplace le caractère (reste de divison)
                   dans la chaine

                inc ebx          ; on incrémente la position courante de la
                   chaine

                cmp eax,0        ; test si eax = 0
                jne boucle       ; alors la conversion est fini. sinon on
                   reboucle

                lea esi,[chaine] ; on recupere l'adresse de début de chaine
                dec ebx          ; on decremente ebx pour qu'il corresponde à l'
                   adresse du dernier caractère de chaine

     ; On réutilise le code de la question précédente
suivant:

                mov eax,[ebx]    ; on parcours la chaine grace a l'adresse dans
                   ebx

                push eax         ; Affichage
                call putchar
                add esp,4

                dec ebx          ; on décrémente notre compteur

                cmp esi,ebx      ; on compare l'adresse de ebx et celle de
                   chaine

                jbe suivant      ; Tant que notre compteur est supérieur on
                   continue

                call getchar
                pop edx          ; on restaure les registres
```

```
                pop ecx
                pop ebx
                pop eax
                ret
main        ENDP
END
```

Le fonctionnement du code est expliqué en commentaires. Cependant à l'execution on n'obtient pas le résultat que l'on veut. On voit s'afficher différents caractères ASCII mais pas notre nombre.
En effet notre chaine contient bien nos différents chiffres, mais pas dans un format caractère. Lorsque *putchar* récupère les caractères, il ne semble pas les prendre en compte comme des caractères.

2.3.2 Affichage du nombre

Afin que le nombre s'affiche de manière lisible, on corrigera très légèrement le code précédent. On ajoute simplement une instruction, juste après la division, et avant la récupération du chiffre. Ce qui donne dans le code précédent :

Assembleur

```
[...]
xor edx,edx      ; mise à 0 du registre
div ecx          ; On effectue la divison : le reste sera dans edx et le quotient
    dans eax
add edx,'0'      ; Transformation en caractère
mov [ebx],edx    ;On déplace le caractère (reste de divison) dans la chaine
[...]
```

On effectue donc une simple addition entre le résulat et '0'. On test et on constate bien que, en pratique, l'affichage se fait correctement. La fonction *putchar* prend bien en compte notre chiffre comme étant un caractère.