COMPTE-RENDU

TP numéro 4 [MI01]

21 NOVEMBRE 2014

Moulin Mathieu & Laviolette Etienne

Introduction :

L'objectif principal de ce TP numéro 4 de MI01 est de se familiariser avec l'assembleur et son environnement de développement. A l'aide des outils à notre disposition, comment faire le parallèle avec un programme en C, comment sont agencées les données d'un programme de ce langage. De nombreux rappels sur le mode d'adressage, la taille des adresses, les sauts conditionnels, éléments primordiaux et caractéristiques du langage assembleur constitueront l'essence même de ce TP.

Exercice – 1 : Prise en main de l'environnement de développement

L'objectif de cette première partie fut de prendre en main les outils à notre disposition et nécessaires à la réalisation de nos TP concernant l'IA32 et l'assembleur.

Nous nous sommes familiarisés avec l'utilisation de Visual Studio 2005 et sa manière de fonctionner. L'ajout d'éléments ainsi que la création de projets furent au centre de cette première sous partie.

Ensuite la structure d'un fichier source a été passée au crible. Des nombreux rappels sur la composition de chaque section d'un fichier source : en-tête, .DATA, .CODE nous ont permis de resituer les éléments constituants d'un programme, leur place l'ordonnancement de ceux-ci lors de la compilation à proprement parler et de l'édition des liens d'un banal programme.

La dernière phase de la prise en main fut très instructive puisqu'elle nous a permis de saisir les subtilités de la fonction de débogage, primordiale lors du suivi de l'état d'un programme instruction après instruction. L'utilité première des différentes options du débogueur est de suivre l'état du processeur et plus précisément des registres lors de l'exécution du programme.

Exercice – 2 : Affichage de chaines de caractères :

Question 1 :

Puisque nous comparons la variable longueur à un registre du processeur (ebx) de 32 bits alors cette même variable longueur doit être de 32 bits. C'est pourquoi nous avons déclaré l'instruction suivante :

« `longueur DD 21` »

Cette déclaration a été ajoutée dans la section. DATA puisqu'elle correspond à l'initialisation d'une variable statique.
Le 21 a été déterminé en comptant trivialement le nombre de caractère de la chaine à afficher.

Question 2 :

L'objectif de ce programme est d'afficher dans le terminal la chaine de caractère « Bonjour tout le monde ».
Pour ce faire le code suivant a été mise en place :

```
; 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'

            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. Cf cours sur les sous-programmes.
            push    eax             ; Caractère à afficher
            call    putchar         ; Appel de putchar
            add     esp, 4          ; Nettoyage de la pile après appel
            ; Fin de l'appel à putchar

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

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

            ret                     ; Retour au code de démarrage 'C'

main        ENDP

            END
```

Le programme (issu du main() d'une fonction C) utilise localement le registre *ebx*. C'est pourquoi la valeur de ebx est empilée et ce registre remis à zéro après empilement.

Tout d'abord on utilise le registre *eax* comme registre de contrôle d'adresse : il est initialisé à l'adresse de début de stockage de la chaine de caractère. On ne récupère que les 8 bits de poids faible de cette adresse (le déplacement dans le segment de mémoire) grâce à l'ajout de la spécification de la longueur de l'adresse : « byte PTR ».

De plus l'étude de la composition du programme nous amène à déterminer que la fonction « putchar » servira à l'affichage des caractères à l'écran du terminal.

Cette fonction prend un caractère comme paramètre donc un registre sera utilisé par le processeur à cet effet : le registre *eax* . On empile donc un à un les caractères à afficher sur la pile, on appelle putchar.

Une fois l'appel terminé on « nettoie » le registre d'empilement *esp* en l'incrémentant de 4 : la taille du paramètre sur la pile.

L'incrémentation de *ebx* assimilable à un registre de boucle et la comparaison à la longueur total de la chaine permet de mettre en place une boucle de type JNE testant les drapeaux du processeur qui ont été (ou pas…) mis en place grâce à l'instruction *cmp.*

Si le drapeau n'est pas mis en place alors on boucle sur l'étiquette de début de boucle « suivant ».

Dans le cas contraire, on est arrivé au bout de la chaine à afficher. On « pop » le registre ebx pour que ebx pointe sur l'instruction suivant l'appel de cette fonction main(), on retourne « ret » puis on signale la fin du main.

L'algorithme utilisé est le suivant :

Début algo_affichage_chaine_de_caractere

```
Initialisation msg : variable de 8 bits  : « Bonjour tout le monde » [type chaine de cacatère]
Initialisation longueur : variable de 8 bits : « 21 » [type entier]
Empilage ebx sur esp  // stockage de sa valeur pré-programme
Ebx <- 0
Pour ebx = 0 jusqu'à ebx = longueur faire :
        Empilage sur eax d'un caractère
Putchar sur eax
Incrémente l'adresse de la pile ESP de 4 // élimination de la donnée traitée
Ebx <- ebx +1
        FinPour

        Getchar de rien // attente de l'appuie sur une touche

        Rétablie ebx à son état pré-main()
Fin_algo_affichage_chaine_de_caracteres
```

Question 3 : Elimination d'une instruction redondante

Optimisation : hello1op.asm

Le programme précédent n'est pas le plus optimisé possible dans la réalisation de la tâche qui est la sienne. C'est pourquoi le hellolop suivant a été codé :

```
; hello1op.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'

            mov     ebx, [longueur] ; on charge la longueur dans ebx
```

```
            lea esi, [msg + ebx] ;resultat du calcul d'adresse de msg+ebx pour se
placer à la fin de la chaine

            neg           ebx ; on met ebx à -ebx a chaque fois on affiche le
caractere a la postion fin -ebx jusqu'a ce qu'on soit a 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 + esi]

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

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

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

            ret                     ; Retour au code de démarrage 'C'

main        ENDP

            END
```

Lors de l'étude du programme précédent on se rend compte d'une redondance dans l'utilisation des drapeaux. En effet si on décide de boucler de -21 à 0 en prenant la négation de longueur (instruction neg en assembleur). L'instruction d'incrémentation inc ebx à chaque tour de boucle positionnera le drapeau « 0 » lorsque ce registre prendra la valeur « 0 ». C'est pourquoi l'instruction « cmp » de notre programme précédent est redondant avec une utilisation plus fine du registre de boucle ebx.

Le pendant de cette utilisation est de chargé grâce à « lea » l'adresse de la fin de la chaine dans esi.

Ainsi le cumul de cette adresse de fin de chaine avec ebx qui démarre à -21 nous permet de parcourir la chaine dans le bon sens. D'afficher à chaque fois le caractère et de sortir de la boucle lorsqu'ebx vaut 0 grâce au même JNE sur l'étiquette suivant du programme précédent

Question 4 : Chaine de taille variable :

```
; hello2.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",0

.CODE

; Sous-programme main, automatiquement appelé par le code de
; démarrage 'C'
PUBLIC      main
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. Cf cours sur les sous-programmes.
            push    eax             ; Caractère à afficher
            call    putchar         ; Appel de putchar
            add     esp, 4          ; Nettoyage de la pile après appel
            ; Fin de l'appel à putchar

            inc     ebx             ; Caractère suivant
            cmp     [ebx+msg], 0 ; Toute la longueur ?
            jne     suivant         ; si non, passer au suivant

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

            ret                     ; Retour au code de démarrage 'C'

main        ENDP

            END
```

Le programme ci-dessus, modification du hello1.asm permet de se passer de la longueur de la chaine « longueur » lors de déroulement du programme.

La comparaison servant de critère d'arrêt à notre boucle est désormais basée sur le fait que le dernier caractère indiquant la fin de la chaine est « 0 ». Donc si le contenu de l'adresse du

registre de boucle (ebx) addition à l'adresse mémoire du début de la chaine (msg) est 0, le drapeau « 0 » se met en place et donc la sortie de boucle s'effectue.

Dans le cas présent, c'est lorsque ebx vaudra 21 que le calcul de l'adresse de msg + 21 aboutira à une adresse contenant 0 donc au positionnement du drapeau et à la sortie de boucle.