TP04 - Prise en main de l'environement de développement assembleur et premiers programmes

Geoffrey PERRIN
Océane DUBOIS

0.1 Exercices : affichage de chaîne de caractère

0.1.1 Programme "Hello World"

Dans le fichier "hello1.asm", la variable msg est définie comme un suite de bytes contenant "bonjour tout le monde". La variable longueure est définie comme un data double word (il occupe ainsi tout un registre) contenant la valeur 21 soit le nombre de caractères (occupant chacun un byte) de la chaîne de caractère msg.

Le programme fonctionne donc ainsi :

- La fonction main (seule fonction du programme), commence par réaliser une sauvegarde pour le code 'C' grâce à push ebx (le contenu de ebx est mis sur le sommet de la pile)
- puis le registre ebx est mis à 0
- puis on définie la ligne de programme qui suit avec l'étiquette "suivant". Cet ligne permet de mettre dans eax le contenu de l'adresse pointée par la somme de ebx et l'adresse de message. Le reste du registre sera completé par des zéros. Ebx va ici nous servir de controleur pour le nombre d'itération effectuées.
- puis on met sur la pile le caratère à afficher et on appel la fonction c "putchar" qui elle-meme appelle fputchar qui permet d'afficher le caractère qui à été mis sur la pile.
- puis on ajoute 4 au pointeur de pile pour le déplacer sur un emplacement non encore utilisé par le programme
- on incrémente ensuite notre itérateur ebx
- on effecture la comparaison entre ebx et la valeur contenue dans la variable longueure, les drapeaux sont mis à jour
- si ebx et la valeur de longeur ne sont pas égaux cela signifie qu'on est pas arrivé à la fin de la chaine de caractère, on retourne donc à la ligne du programme ayant comme étiquette suivante et on réalise cette boucle jusqu'a ce que la valeur de longueure soit égale à la valeur de ebx.
- si ebx et le contenu de longueur sont égaux on peut appeler la fonction c getchar qui attent l'appui sur "Entrée"

- on met le sommet de la pile dans ebx
- on signal au programme qu'il peut retourner au code de démarrage 'C' puis on fini le programme avec l'instruction "main endp"

Le fonctionnement de cet algorithme pourrait ressembler au fonctionnement d'un programme contenant l'instruction "while(ebx != longueur) en C. Et comme déjà dit précédement ebx nous sert d'itérateur (et donc la condition d'arrêt dépent de lui).

```
.model      flat, c

extern      putchar:near
extern      getchar:near

.data

msg db "bonjour tout le monde"
longueur dd 21
; Ajoutez les variables msg et longueur ici

.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]

            ; 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
            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'
) %
rtie
```

Figure 1: Programme "hello1"

0.1.2 Chaîne de taille variable

On modifie maintenant le premier code en "hello2" dans lequel la variable longueure n'est plus utilisée, à la place on définie la variable msg comme la chaine de caractère "bonjour tout le monde", terminée par un 0.

Nous avons donc modifié le programme pour qu'il affiche toujours correctement le contenu de msg. Pour cela on garde ebx qui nous sert toujours de variable permettant de parcourir chaque caractère du message à afficher. Mais la comparaison se fait entre eax et 0 en effet si eax est égal à 0 cela signifie qu'on est arrivé à la fin du message (cela est valable car la chaîne message ne contient pas de caractère 0).

L'algorithme implémenté correspond donc à un while(msg[ebx] != 0)

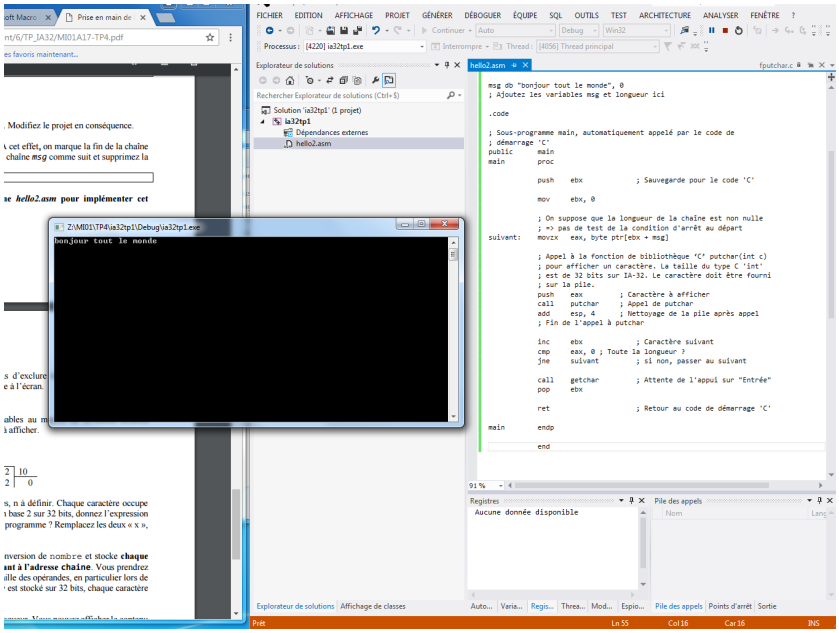

Figure 2: Programme "hello2"

0.2 Exercice : conversion et afficahge de nombres

0.2.1 Conversion d'un nombre non signé en base 10

Dans ce programme on cherche à convertir et afficher des nombres en différentes bases. Pour cela on utiliser une chaine de caractère de taille n pour stocker la conversion de nombre. Sachant nombre exprimé en base 2 sur 32 bits. Sa valeur maximale serat donc FFFFFFFFh soit 2^{32}.
Nous souhaitons convertir le tout en base 10, on résout donc $2^{32} = 10^n$.
$2^{32} = 10^n$
$32 * ln(2) = n * ln(10)$

$n = \frac{32 * ln(2)}{ln(10)}$

$n = 9.63295986125$

$\lceil n \rceil = 10$

On aura donc besoin d'au plus 10 caractères pour afficher un nombre de 32bits en décimal.

Pour convertir on utilisera la méthode des divisions successive qui fonctionne pour les entiers non signés. On divise donc successivement l'entier par la base de conversion, on stocke les restes et on divise le résultat jusqu'à ce que le resultat de la division soit égale a 0.

```
1  title conversion.asm
2
3  .686
4  .model    flat, c
5
6  extern      putchar:near
7  extern      getchar:near
8
9  .data
10
11 nombre      dd      95c8ah          ; Nombre a convertir
12 chaine      db      10 dup(?)       ; longueur maximale n de la
       chaine
13
14 .code
15
16 ; Sous-programme main, automatiquement appele par le code de
17 ; demarrage 'C'
18 public      main
19 main        proc
20       push  eax             ; sauvegarde des registres
21       push  ebx             ; Sauvegarde pour le code 'C'
22
23       xor   ebx, ebx
24       xor   eax, eax
25       mov   ecx, 10         ;base dans laquelle on souhaite
       convertir la variable nombre
26       mov   eax, [nombre]
27
28 suivant :
29       xor   edx, edx
30       div     ecx       ; on divise eax (nombre) par ecx (ici 10)
31       mov     [chaine+ebx], dl ; le reste est placé dans le
       registre edx (le reste sera toujours inferieur a 32 donc il
```

```
    occupera donc toujours au plus les 4 premiers bits de dl)
    chaque case de chaine étant de 1 octet on copie le registre
    dl (de 1 octet aussi) dans notre case mémoire de chaine
32      inc     ebx             ; on incrémente ebx pour qu'a la
    prochaine occurence, dl soit placé dans la case mémoire
    suivante
33      cmp     eax, 0          ; si le resultat de la division (
    registre eax) est 0 alors la conversion est terminé sinon on
    continue
34      jne     suivant
35
36      call    getchar         ; Attente de l'appui sur "Entree"
37      pop     ebx
38      ret             ; Retour au code de demarrage 'C'
39
40  main        endp
41
42  end
```

La chaine contient les chiffres brut. Mais putchar affiche les caractères avec la table ASCII.
Le caractere 0 ne correspond pas au chiffre 0 dans la table ASCII. On ajoute donc "O" (soit le numero du caractere "0" dans la table ASCII) a nos chiffres pour les faire correspondre aux bon caractere et donc les afficher correctement.

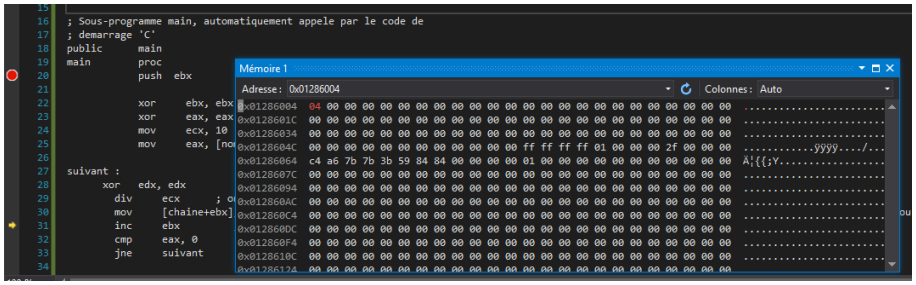

Figure 3: Etat de la mémoire après la première iteration

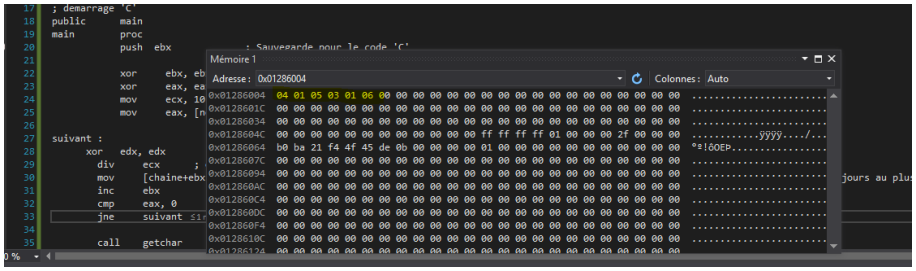

Figure 4: Etat de la mémoire a la fin de l'itération

0.2.2 Affichage du nombre

```
; Sous-programme main, automatiquement appele par le code de
; demarrage 'C'
public      main
main        proc

      push  ebx             ; Sauvegarde pour le code 'C'

      xor   ebx, ebx
      xor   eax, eax
      mov   ecx, 10
      mov   eax, [nombre]

suivant :
      xor   edx, edx
      div     ecx
      mov   [chaine+ebx], dl
      inc   ebx
      cmp   eax, 0
      jne   suivant
      dec   ebx     ; on reprend le programme précédent et on
   decremente ebx pour avoir le bon nombre de chiffres de notre
   nombre converti.


affichage :

      movzx   eax, byte ptr[ebx + chaine]  ; on met dans le
   registre eax un cara caractere de chaine a chaque iteration.
   On commence par la fin (ici ebx est égale l'index du dernier
   caractere de chaine)

      add     eax, "0"        ; on ajoute le numero du caractere
    '0' a notre chiffre pour l'afficher correctement
      push    eax             ; on ajoute sur la pile le
```

```
    caractere a afficher
29      call    putchar         ; Appel de putchar
30      add     esp, 4          ; Nettoyage de la pile apres appel
31      dec     ebx             ; Caractere precedent
32      cmp     ebx, 10         ; on compare ebx a 10. Quand ebx
    est égale a 0 on est au debut de la chaine on a donc affiche
    le dernier caractere.
33      jb      affichage       ; si on decremente encore ebx sera
     égale a FFFFFFFFh. Le nombre maximum que peut contenir ebx
    est le nombre maximum de caractere (ici 10, si il est
    superieur a 10 c'est qu'il est égale a FFFFFFFFh)
34      call    getchar         ; Attente de l'appui sur "Entree"
35      pop     ebx
36      ret                     ; Retour au code de demarrage 'C'
37
38  main    endp
39
40  end
```

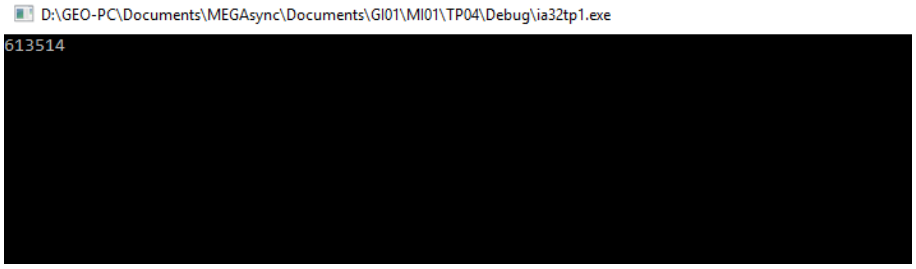

Figure 5: Execution de la conversion

0.2.3 Affichage du signe

La méthode actuellement implémentée ne fonctionne pas pour les nombres négatifs. Pour qu'elle fonctionne nous devons d'abord réaliser un test permettant de déterminer si le nombre est négatif ou non. Si il l'est,on affiche le caractere "-", on prendre son opposé et on lance le programme déjà conçu précédement

```
1  .data
2
3  nombre      dd      -95c8ah          ; Nombre a convertir
4  chaine      db      32 dup(?)       ; Remplacer xx par la
    longueur maximale n de la chaine
5
6  .code
```

```
7
8  ; Sous-programme main, automatiquement appele par le code de
9  ; demarrage 'C'
10 public      main
11 main        proc
12
13       push  ebx             ; Sauvegarde pour le code 'C'
14
15       xor   ebx, ebx
16       xor   eax, eax
17       mov   ecx, 10
18       mov   eax, [nombre]
19       cmp   eax, 0          ;on compare le nombre a 0
20       jge   suivant         ;si il est positif ou nul on ne fait
     rien et on passe a la conversetion directement
21       dec   eax             ;si le nombre est negatif on le
     converti en son oppose en decrementant 1
22       not   eax             ;et avec l'operation not
23       push  eax             ;sauvegarde des registres eax et ecx
     avant l'appel de putchar
24       push  ecx
25       push  "-"             ; on affiche le - car le nombre est
     negatif
26       call    putchar       ; Appel de putchar
27       add     esp, 4        ; Nettoyage de la pile apres appel
28       pop ecx               ; on recupere les registres eax et ebx
      sauvegardé dans la pile
29       pop eax
30
31 suivant :
32       xor   edx, edx
33       div ecx
34       mov   [chaine+ebx], dl
35       inc   ebx
36       cmp   eax, 0
37       jne   suivant
38       dec   ebx
39
40
41 affichage :
42       movzx   eax, byte ptr[ebx + chaine]
43       add     eax, "0"
44       push    eax             ; Caractere a afficher
45       call    putchar         ; Appel de putchar
46       add     esp, 4          ; Nettoyage de la pile apres appel
47       dec     ebx             ; Caractere suivant
48       cmp     ebx, 10         ; Toute la longueur ?
49       jb      affichage       ; si non, passer au suivant
50       call    getchar         ; Attente de l'appui sur "Entree"
```

```
51      pop     ebx
52      ret                     ; Retour au code de demarrage 'C'
53
54  main        endp
55
56  end
```

Figure 6: Execution avec -95c8ah

0.2.4 Base quelconque entre 2 et 36

On veut donc pouvoir convertir un nombre dans n'importe quelle base entre 2 et 36, pour cela il suffit de réaliser la division par cette base. De plus on n'utilisera pas la table ASCI mais une chaîne de caractères composée des caractères allant de 0 à 9 puis de A à Z. Cette chaîne contient donc tous les digit possibles pour écrire des nombes allant de la base 2 à la base 36. Pour déterminer quel caractère il faut afficher on utilisera un pointeur sur cette chaine de caractère qui se déplacera en fonction du résultat de la division.

On aura besoin au maximum d'afficher 32 caractères, la chaîne doit donc avoir pour longueur 32 octets.

```
1  ; conversion.asm
2  ;
3  ; MI01 - TP Assembleur 1
4  ;
5  ; Affiche un nombre de 32 bits sous forme lisible
6
7  title conversion.asm
8
9  .686
10 .model    flat, c
```

```
11
12 extern       putchar:near
13 extern       getchar:near
14
15 .data
16
17 nombre       dd       95c8ah          ; Nombre a convertir
18 chaine       db       32 dup(?)       ; Remplacer xx par la
       longueur maximale n de la chaine
19 chiffres     db       "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20
21 .code
22
23 ; Sous-programme main, automatiquement appele par le code de
24 ; demarrage 'C'
25 public       main
26 main         proc
27       push   ebx             ; Sauvegarde pour le code 'C'
28
29       xor    ebx, ebx
30       xor    eax, eax
31       mov    ecx, 35
32       mov    eax, [nombre]
33       cmp    eax, 0
34       jge    suivant
35       dec    eax
36       not    eax
37       push   eax             ; sauvegarde des registres
38       push   ecx
39       push   "-"             ; on affiche le - si le nombre est
       negatif
40       call     putchar       ; Appel de putchar
41       add      esp, 4        ; Nettoyage de la pile apres appel
42       pop ecx
43       pop eax
44
45 suivant :    xor    edx, edx
46       div    ecx
47       mov    [chaine+ebx], dl
48       inc    ebx
49       cmp    eax, 0
50       jne    suivant
51       dec    ebx
52
53
54 affichage : movzx    eax, byte ptr[ebx + chaine]
55
56       lea      edx, [chiffres + eax]
57       push     [edx]           ; Caractere a afficher
```

```
58          call    putchar             ; Appel de putchar
59          add     esp, 4              ; Nettoyage de la pile apres appel
60          dec     ebx                 ; Caractere suivant
61          cmp     ebx,32              ; Toute la longueur ?
62          jb      affichage           ; si non, passer au suivant
63          call    getchar             ; Attente de l'appui sur "Entree"
64          pop     ebx
65          ret                         ; Retour au code de demarrage 'C'
66
67  main        endp
68
69  end
```

En base 35, 95c8ah est égale à EASY

Figure 7: Resultat final