Leonardo D'ALESSIO - GI01
Soulaymane KEBLI - GI01

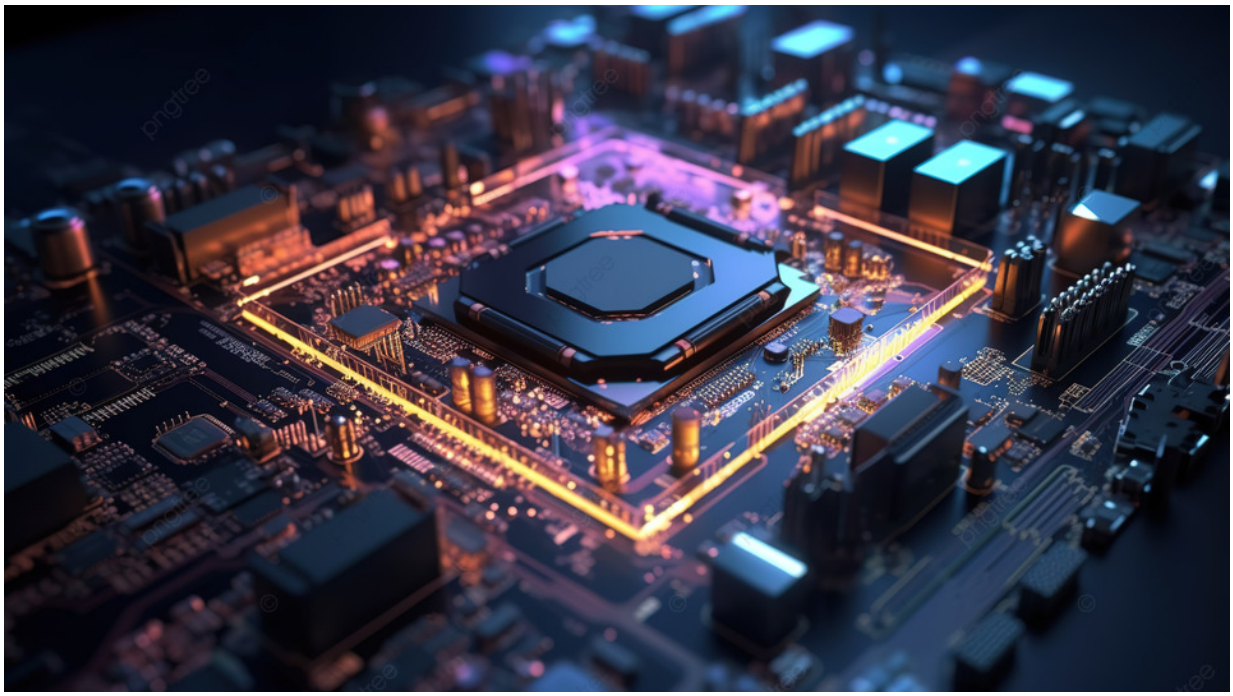
Université de Technologie de Compiègne

MI01 - Structure d'un calculateur

Responsable
Mr. Shawky Marc

TP4 - ASSEMBLEUR ET, PREMIERS PROGRAMMES

Responsable TP
Mr. Sanahuja Guillaume



TP1 - Groupe n°6

Sommaire

Table des figures

1 Exercice 1 - Affichage de chaînes de caractères

1.1 Partie A - Programme « Hello world »

L'objectif de cet exercice est de définir une variable *msg*, de type « chaîne de caractères » qui contient « Bonjour tout le monde ! » et une variable *longueur* qui contient la longueur de la chaîne. Le programme doit afficher le contenu de la variable *msg*.

1.1.1 Question 1

R13 étant un registre d'une taille de 64 bits, il est nécessaire d'allouer en mémoire à la variable *longueur* un espace de 64 bits pour rendre la comparaison faisable.
On a déclaré la variable *longueur* de cette manière :

```
.file "hello1.S"
.intel_syntax noprefix

.data

msg:        .ascii  "Bonjour tout le monde!"
longueur:   .quad   22
```

1.1.2 Question 2

Ce programme en langage assembleur x86 affiche le message "Bonjour tout le monde!" (ou du moins la chaîne de caractère stockée dans la variable *msg*) caractère par caractère. Il fonctionne de la manière suivante :

```
On stocke l'adresse de msg dans r12
On stocke la valeur 0 dans r13
Pour r13 allant de 0 à longueur :
    écrire le caractère situé à l'adresse r12+r13 sur la sortie standard,
    incrémenter r13
fin pour
```

Le registre *r13* est utilisé comme un compteur pour parcourir la chaîne de caractères. Il commence à zéro et est incrémenté à chaque itération de la boucle *suivant* jusqu'à ce qu'il atteigne la longueur de la chaîne (*longueur*). Il est également utilisé pour accéder à chaque caractère individuel de la chaîne à afficher.
Afin de décrire le programme, voici une version commentée de la section code du fichier "hello1.S" :

```
.global _start
_start:
    lea     r12, msg[rip]       /* Adresse de début de la chaîne dans r12 */
    mov     r13, 0              /* Initialise le compteur de caractères a 0 (r13) */


    mov     rdi, 1    /* Charge stdout (1) dans le registre rdi pour l affichage */
    mov     rdx, 1              /* 1 seul caractère a afficher a la fois */
```

```
suivant:                          Boucle pour afficher les caractères
    lea     rsi, [r12+r13]      /* Adresse du caractère a afficher */
    mov     rax, 1              /* Appel système : ecrire (write) */
    syscall

    add     r13, 1              /* Passage au caractère suivant dans la chaîne */
    cmp     r13, longueur[rip]  /* Comparaison avec la longueur de la chaîne */
    jb      suivant             /* Tant que r13 < longueur, continue */

fin:
    mov     rdi, 0              /* Valeur de retour (0) */
    mov     rax, 60             /* Appel système : exit */
    syscall                     /* Termine le programme */
```

1.1.3 Question 3

Après avoir placé notre line breakpoint à la ligne 27 (début du programme), on lance le programme en mode debug en effectuant chaque instruction une par une (pas à pas).

```
25 .global _start
26 _start:
27      lea     r12, msg[rip]
28      mov     r13, 0
29
30      mov     rdi, 1              /* fd = 1 (stdout)                    */
31      mov     rdx, 1              /* 1 seul caractère                   */
32 suivant:
33      lea     rsi, [r12+r13]      /* Adresse du caractère               */
34      mov     rax, 1              /* Appel no 1 (write)                 */
35      syscall                     /* Appel système                      */
36
37      add     r13, 1              /* Passer au caractère suivant        */
38      cmp     r13, longueur[rip]  /* Toute la longueur ?                */
39      jb      suivant             /* Si non, passer au suivant          */
40
41      /* Terminer le processus.
42         On utilise l'appel système exit(retval) pour mettre fin à un
43         processus. Cet appel est le nunméro 60.                        */
44      mov     rdi, 0              /* Valeur de retour (ici 0)           */
45      mov     rax, 60             /* Appel  60 (exit)                   */
46      syscall
```

FIGURE 1 – *hello1.S en mode Debug*

Console | Registers | Memory | Problems | Executables | Memory Browser | Debugger Console

Name	Value
General Registers	
rax	1
rbx	0
rcx	93824992235561
rdx	1
rsi	93824992243714
rdi	1
rbp	0x0
rsp	0x7fffffffdf80
r8	0
r9	0
r10	0
r11	919
r12	93824992243712
r13	2
r14	0
r15	0

FIGURE 2 – *Registres hello1.S en mode Debug*

On peut observer grâce au débogueur que les valeurs des registres varient en fonction des itération de la boucle et qu'on voit bien la chaîne s'afficher lettre par lettre.

1.2 Partie B - Chaîne de taille variable

On veut maintenant se passer de la longueur de la chaîne à afficher. À cet effet, on marque la fin de la chaîne de caractères par un caractère NUL (code ASCII 0). Pour cela on supprime la variable *longueur* et on modifie le type de la variable *msg*. En effet, on lui attribue cette fois le type *.asciz* pour décrire une chaîne de caractères terminée par un caractère NUL. La section de donné du fichier "hello2.S" se définit comme suit :

```
.file "hello2.S"
.intel_syntax noprefix

.data

msg:        .asciz        "Bonjour tout le monde!"
```

1.2.1 Question 1

Contrairement à la partie A, nous n'avons pas de variable *longueur* sur laquelle nous pouvons nous appuyer pour créer une boucle à bornes définies en algorithmie. Nous allons donc créer une boucle à borne indéfinies avec comme condition d'arret la comparaison entre le dernier caractère et la valeur ASCII 0.
L'algorithme se présente de cette manière :

```
On stocke l'adresse de msg dans r12
On stocke la valeur 0 dans r13
On stocke la vlaeur 0 dans r8
Tant que rsi <> r8 :    # = tant que le caractère est non nul
    écrire le caractère situé à l'adresse r12+r13 sur la sortie standard,
    incrémenter r13,
fin tant que
```

1.2.2 Question 2

Voici le code du fichier "hello2.S" :

```
.global _start
_start:
        lea     r12, msg[rip]
        mov     r13, 0
        mov     r8, 0

        mov     rdi, 1              /* fd a 1 (stdout) */
        mov     rdx, 1              /* 1 seul caractère */
suivant:
        lea     rsi, [r12+r13]      /* Adresse du caractère */
        mov     rax, 1              /* Appel no 1 (write) */
        syscall                     /* Appel système */

        add     r13, 1              /* Passer au caractère suivant */
        cmp     r8, [rsi]     /* Compare r8 (0) et le contenu de rsi (caractere) */
        jne     suivant           /* Si non, passer au suivant*/
```

```
fin:
        mov     rdi, 0              /* Valeur de retour (ici 0) */
        mov     rax, 60             /* Appel  60 (exit) */
        syscall
```

2 Exercice 2 : Conversion et Affichage de nombres

L'objectif de l'exercice est d'écrire un programme qui construit une chaîne de caractères contenant la représentation d'un nombre et qui l'affiche sur la console, d'abord en base 10, puis dans une base quelconque.

2.1 Partie A - Conversion d'un nombre non signé en base 10

2.1.1 Question 1

La formule permettant de calculer le nombre de caractères pour représenter un nombre de 64bit en décimale est :

$$n > \log_{10}(2^{64} - 1) \iff n > 19.3 \implies On\ prend\ n = 20$$

On remplace alors les points d'interrogations par 20. La section de données du programme se présente comme ci-dessous :

```
.file "conversion.S"
.intel_syntax noprefix

.data

nombre:         .quad   0x0451faf7d3c41971
chaine:         .fill   20
```

2.1.2 Question 2

Voici la section code commentée du programme réalisé dans le fichier "conversion.S" :

```
.global _start
_start:
    xor r13,r13             /* remise a 0 de r13 (compteur pour la chaîne) */
    mov rax, nombre[rip]    /* Charge le nombre dans rax */
    mov rbx, 10             /* Charge 10 dans rbx (pour la division) */
    lea r12, chaine[rip]    /* Adresse de début de la chaîne dans r12 */

boucle:
    xor rdx, rdx            /* Efface rdx (quotient de la division précédente) */
    div rbx                 /* Divise rax par rbx (nombre / 10) */
    add dl, 0x30            /* Convertit le reste en caractère ASCII */
    mov [r12 + r13], dl     /* Stocke le caractère converti dans la chaîne */
    add r13, 1              /* Incrémente le compteur pour la chaîne */
    cmp rax, 0              /* Compare le quotient avec 0 */
    jne boucle              /* Continue la boucle si rax different de 0 */
```

Dans ce programme, le registre *r13* est utilisé comme compteur pour la chaîne de caractères, initialisé à zéro pour suivre la position actuelle dans la chaîne à remplir.
Le nombre hexadécimal stocké dans *nombre* est chargé dans le registre *rax*, tandis que la valeur 10 est chargée dans *rbx*, servant de diviseur pour extraire les chiffres du nombre.
La boucle *boucle* commence par diviser le nombre dans *rax* par 10 (*div rbx*). Le reste de cette division (stocké dans *dl*) est converti en caractère ASCII représentant le chiffre et est placé dans la chaîne à l'adresse pointée par *r12 + r13*. Ensuite, le compteur *r13* est incrémenté pour pointer vers la prochaine position dans la chaîne.

Cette opération se répète jusqu'à ce que le quotient de la division devienne zéro (*cmp rax, 0*). Cela signifie que tous les chiffres du nombre ont été extraits et convertis en caractères ASCII.

Pour vérifier le fonctionnement de notre programme on fait appel au débogueur comme dans l'exercice 1 :

```
21 .global _start
22 _start:
23         /*****************************************************************
24          Début de votre code
25          *****************************************************************/
26         xor r13,r13            /* remise à 0 de r13 */
27         mov rax, nombre[rip]
28         mov rbx, 10
29         lea r12, chaine[rip]
30
31 boucle:
32         xor rdx, rdx
33         div rbx
34         add dl, 0x30               /* pour convertir en caractère ASCII, on ajoute + 0x30 */
35         mov [r12 + r13], dl
36         add r13, 1
37         cmp rax, 0
38         jne boucle
```

FIGURE 3 – *conversion.S en mode Debug*

Console | Registers | Memory | Problems | Executables | Memory Browser | Debugger Console

Name	Value	Des
General Registers		Ge
rax	311305791	
rbx	10	
rcx	0	
rdx	53	
rsi	140737354131232	
rdi	140737354129792	
rbp	0x0	
rsp	0x7fffffffdf30	
r8	0	
r9	0	
r10	0	
r11	518	
r12	93824992243720	
r13	9	
r14	0	
r15	0	

FIGURE 4 – *Registres conversion.S en mode Debug*

On peut observer grâce au débogueur que les valeurs des registres varient en fonction des itération de la boucle. Le programme semble alors correctement fonctionner.

2.2 Partie B - Affichage du nombre

On ajoute au code précédent une nouvelle partie *afficher* qui est une boucle permettant d'afficher à l'écran la chaîne obtenue.

```
.global _start
_start:
        xor r13,r13             /* remise a 0 de r13 (compteur pour la chaîne) */
        mov rax, nombre[rip]    /* Charge le nombre dans rax */
        mov rbx, 10             /* Charge 10 dans rbx (pour la division) */
        lea r12, chaine[rip]    /* Adresse de début de la chaîne dans r12 */

boucle:
        xor rdx, rdx            /* Efface rdx (quotient de la division précédente) */
        div rbx                 /* Divise rax par rbx (nombre / 10) */
        add dl, 0x30            /* Convertit le reste en caractère ASCII */
        mov [r12 + r13], dl     /* Stocke le caractère converti dans la chaîne */
        add r13, 1              /* Incrémente le compteur pour la chaîne */
        cmp rax, 0              /* Compare le quotient avec 0 */
        jne boucle              /* Continue la boucle si rax different de 0 */

                                Préparation pour l appel système
        mov rdi,1               /* rdi a 1 (pour stdout) */
        mov rdx,1               /* rdx a 1 (1 caractère a ecrire) */
        mov rax,1               /* Appel système 1 (write) */


afficher:
        sub r13, 1              /* Décrémente r13 pour pointer vers le dernier
                                elément de la chaîne*/
        lea rsi, [r12+r13]      /* Prépare l adresse de l element a
                                afficher(début de la chaîne)*/
        syscall                 /* Appel système pour afficher le caractère actuel */
        cmp r13, 0              /* Compare r13 avec zéro pour vérifier si
                                toute la chaîne a ete parcourue */
        jne afficher            /* Si r13 n est pas egal a zero, saute a
                                afficher pour continuer l affichage */


fin:
        mov     rax, 60             /* Appel 60 (exit)*/
        mov     rdi, 0              /* Valeur de retour (ici 0)*/
        syscall
```

Une fois la conversion terminée, le programme est prêt pour un appel système afin d'écrire la chaîne résultante caractère par caractère sur la sortie standard (stdout).

Le label *afficher* marque le début de l'affichage. La première instruction ; *sub r13, 1*, décrémente *r13* pour pointer vers le dernier élément de la chaîne générée. Cela permet de commencer la lecture inverse depuis la fin de la chaîne.
En utilisant l'instruction *lea rsi, [r12+r13]*, le programme prépare l'adresse de l'élément

actuel à afficher en se déplaçant depuis l'adresse de début de la chaîne (r12) jusqu'à l'élément pointé par r13.
L'appel système syscall est ensuite utilisé pour afficher le caractère actuel de la chaîne. Ce processus de lecture inverse est répété pour chaque caractère de la chaîne.

La boucle se poursuit avec les instructions *cmp r13, 0* et *jne afficher*. La comparaison cmp vérifie si *r13* est égal à zéro, indiquant la fin de la chaîne. Si *r13* n'est pas nul, le programme saute à l'étiquette lecture pour continuer la lecture inverse.

2.3 Partie C - Base quelconque entre 2 et 36

2.3.1 Question 1

Afin de pouvoir convertir le reste de la division en caractère affichable nous devons procéder de la manière suivante :

```
.file "conversion_C.S"
.intel_syntax noprefix

.data

nombre:         .quad   0x0451faf7d3c41971
chaine:         .fill   20
chiffres:       .ascii  "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"

.global _start

_start:
        xor r13,r13
        mov rax, nombre[rip]
        mov rbx, 10           /*mov rbx, 29  SI CONVERSION EN BASE 29*/
        lea r12, chaine[rip]
        lea r11, chiffres[rip]

boucle:
        xor rdx, rdx
        div rbx
        mov dl, [r11+rdx]      /* on met dans dl la valeur ascii correspondant
                                     a la valeur de l adresse du tableau plus
                                     le reste de la division */
        mov [r12 + r13], dl
        add r13, 1
        cmp rax, 0
        jne boucle
```

En effet, *rdx* étant le registre ou est stocké le reste de la division entière, et *r11* celui où est stocké l'adresse du tableau de caractère, si le reste de la division nous donne n, nous devons prendre le n-ème caractère.

2.3.2 Question 2

La taille de la chaîne varie selon la base utilisé. En effet, la taille de la chaîne croît pendant que la base décroît. La plus petite base est la base 2, donc il nous faudra au maximum 64 caractères.
Voici le code complet et modifié :

```
.file "conversion_C.S"
.intel_syntax noprefix

.data

nombre:         .quad   0x0451faf7d3c41971
chaine:         .fill   64
chiffres:       .ascii  "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"

.text

.global _start
_start:
                xor r13,r13
                mov rax, nombre[rip]
                mov rbx, 10             /*mov rbx, 29  SI CONVERSION EN BASE 29*/
                lea r12, chaine[rip]
                lea r11, chiffres[rip]

boucle:
                xor rdx, rdx
                div rbx
                mov dl, [r11+rdx]
                mov [r12 + r13], dl
                add r13, 1
                cmp rax, 0
                jne boucle

                mov rdi,1               /* préparation du syscall */
                mov rdx,1
                mov rax,1

afficher:
                sub r13,1
                lea rsi, [r12+r13]
                syscall
                cmp r13, 0
                jne afficher

fin:
        mov     rax, 60             /* Appel 60 (exit) */
        mov     rdi, 0              /* Valeur de retour (ici 0) */
        syscall
```

2.3.3 Question 3

À l'aide de notre programme, nous pouvons maintenant calculer la conversion du nombre **0x0451faf7d3c41971** écris en hexadécimal :

- En base 10 : **311305791581985137**
- En base 29 : **PERMPROFS712**