Université Gustave Eiffel

ESIEE PARIS

Architecture des ordinateurs
TP 3 - CPU 80386

Halim Djerroud

révision 1.0

Exercice 1 :

1. Créer un fichier "printf_example.s" et copier le code suivant :

```
    .data
str1:   .asciz      "Hello Mon premier programme en assembleur \n"
str2:   .asciz      "La valeur de eax = %d \n"

    .text
    .global main

main:
        pushl   %ebp
        movl    %esp,  %ebp

        /* Imprimer str1 en utilisant la fonction printf de la libc */
        pushl   $str1
        call    printf

        /* Utiliser printf avec un parametre  */
        movl     $42, %eax
        pushl    %eax
        pushl    $str2
        call     printf
        addl     $4, %esp
        leave
        ret
```

   Compilez et exécuter le programme, si vous utilisez gcc en ligne de commande au lieu de nasm, ci-après les commandes pour compiler votre programme en 32 bits :
   `gcc -m32 -no-pie printf_example.s -o printf_example`
   Attention : l'option -m32 permet d'utiliser l'architecture 32 bits sur une machine 64 bits. Pour exécuter votre programme :
   `./printf_example`
2. Écrire un programme en assembleur qui affiche à l'écran un le message suivant : "L'assembleur est cool ! " en utilisant la fonction de la librairie standard `printf`.
3. Écrire une programme en assembleur qui affiche les nombre de 1 à 10
4. Écrire une programme en assembleur qui affiche les nombres paires de 1 à 20
5. Écrire une programme en assembleur qui affiche les nombres premiers entre 2 à 100

Exercice 2 : Les sous-programmes (les fonctions)

1. Reprendre le dernier exercice de la partie précédente. Séparer votre programme en deux parties :
   (a) Une fonction qui prend en paramètre un entier et retourne en (0 ou 1) pour indiquer si l'entier donné est premier ou pas

(b) une seconde fonction qui parcoure tous les entiers impairs de (3 à 199) et affiche uniquement les entiers premier, en utilisant la fonction précédente

Exercice 3 : Les modes d'adressage et les tableaux

Les tableau :

1. Écrire une fonction `print_tab` qui prend en paramètre l'adresse d'un tableau d'entiers et sa taille et affiche l'ensemble de ses éléments sous la forme suivante : $[e_1, e_2, ..., e_n]$
2. Écrire une fonction `read_tab` qui prend en paramètre l'adresse d'un tableau d'entiers et sa taille et demande à l'utilisateur de saisir les valeurs (astuce : utilisez `printf` et `scanf`)
3. Écrire une fonction `sum_tab` qui prend en paramètre l'adresse d'un tableau d'entiers et sa taille et retourne la somme des ses éléments
4. Écrire une fonction qui prend en paramètre l'adresse d'un tableau d'entiers et sa taille et retourne la position du plus petit élément
5. Écrire une fonction qui prend en paramètre l'adresse d'un tableau d'entiers et sa taille et inverse ses éléments. Exemple :

```
Tableau en entrée : [1,5,7,8,2,4,8]
Tableau inversé   : [8,4,2,8,7,5,1]
```

6. Écrire une fonction qui retourne la position de la première occurrence d'une valeur, si elle existe, dans un tableau.
7. Écrire une fonction qui retourne **affiche** nombre d'occurrence d'une valeur dans un tableau ainsi que les positions où elle apparaît dans un tableau
8. Écrire une fonction qui teste si un tableau est ordonné
9. Écrire une fonction qui insère une valeur dans un tableau trié
10. Écrire une fonction qui décale les valeurs dans un tableau d'une position vers la gauche
11. Dans la section data on déclare la matrice suivante :

```
.data
tab:    .int 1,5,9
        .int 4,8,7
```

Écrire un programme qui permet de parcourir la matrice `tab` en utilisant deux boucles imbriquées

Les chaînes de caractères :

1. Écrire une fonction `str_len` qui permet trouver le nombre de caractères dans une chaîne de caractères (la taille d'une chaîne).
2. Écrire une fonction `copy_str` qui permet copier une chaîne de caractères dans une autre, les adresses des deux chaînes sont données en paramètre
3. Écrire une fonction qui permet concaténer deux chaînes de caractères.
4. Écrire une fonction `str_cmp` qui permet comparer deux chaînes de caractères `str1` et `str2` selon l'ordre lexicographique. La fonction doit renvoyer `1` si `str1 > str2`, `-1` si `str1 < str2`, `0` sinon.
5. Écrire une fonction qui permet convertir les lettres minuscules dans une chaîne de caractères en lettres majuscules.
6. Écrire une fonction qui `rev_str` permet inverser une chaîne de caractères
7. Écrire une fonction qui permet vérifier si une chaîne de caractères est un palindrome
8. Écrire une fonction qui permet trouver le caractère le plus fréquent dans une chaîne de caractères
9. Écrire une fonction qui calcule le nombre d'alphabets, le nombre de chiffres et le nombre de caractères spéciaux dans une chaîne de caractères. On supposes que les paramètres sont données par référence

Utilisation des conventions

1. Pour chacune des fonctions écrite précédemment (exercice 3) vous devez l'appeler et la tester depuis un programme C.

Exercice 4 : Les appels systèmes

Les appels systèmes sont listé ci-dessus (dans ce document on utilise uniquement les appels 32 bits). La liste complète est fourni dans les documents de cours sans la section *ressources*
ou dans les headers du noyau linux ici :
/usr/include/x86_64-linux-gnu/asm/unistd_32.h

```
#ifndef _ASM_X86_UNISTD_32_H
#define _ASM_X86_UNISTD_32_H 1

#define __NR_restart_syscall 0
#define __NR_exit 1
#define __NR_fork 2
#define __NR_read 3
#define __NR_write 4
#define __NR_open 5
#define __NR_close 6
#define __NR_waitpid 7
#define __NR_creat 8
#define __NR_link 9
#define __NR_unlink 10
....
```

Exemple :

```
/***  DATA SECTION ***/
.data

str:    .asciz "Hello World\n"
        len = . - str - 1

/***  TEXT SECTION ****/

.text
        .global _start

_start:
        /** Appel System **/
        movl $0x4, %eax
        movl $0x1, %ebx
        movl $str, %ecx
        movl $len, %edx
        int $0x80

        /** EXIT **/
        movl $0x1, %eax
        movl $0x0, %ebx
        int $0x80
```

Compilation

```
as -o helloWorld.o helloWorld.s
ld -o helloWorld helloWorld.o
```

Exercice :

1. Écrire un programme en assembleur qui affiche à l'écran un le message suivant : "L'assembleur est cool ! " en utilisant les appels systèmes adéquats.
2. Écrire un programme qui permet de créer un fichier texte vide "test.txt"
3. Écrire un programme qui permet de lire un fichier texte préalablement rempli "myfile.txt".