TD7 : Introduction à l'assembleur sous linux

Première année d'IUT Informatique

1 Premier programme

Voici un exemple

```
.data  #debut segment de donnée
bonjour:
        .string "hello world!\n"
taille:
        .long taille - bonjour  #longueur du texte bonjour
.align 4

.text  #début segment du code
.globl main  #rend le symbolel visible
main:
       ## ecriture via l'appel système write ()
        movl $4, %eax           # write () system call
        movl $1,%ebx            # %ebx = 1, fd = stdout
        leal bonjour, %ecx      # %ecx ---> hello
        movl taille, %edx       # %edx = count
        int $0x80               # execute write () system call

        ##fin de programme via   exit () system call
        xorl %eax, %eax         # %eax = 0
        incl %eax               # %eax = 1 system call _exit ()
        xorl %ebx, %ebx         # %ebx = 0 normal program return code
        int $0x80               # execute system call _exit ()
```

1. Sauvegarder ce texte dans un fichier dont le nom se termine par ".s", par exemple `hello.s`
2. Générer un exécutable avec la commande gcc , exemple : `gcc hello.s -o hello`
3. Vérifier l'exécution :

```
JCB@taxus/TESTS 47 % gcc hello.s -o hello
JCB@taxus/TESTS 48 % hello
hello  world!
```

2 Utilisation d'un "debuggueur"

`gdb` est un debuggueur fourni par GNU (fournisseur de logiciel libre). La commande `man gdb` permet d'afficher le mode d'emploi.

1. Lancer `gdb` sur l'exécutable que vous venez de générer, exemple `gdb hello`
2. Commenter la série de manipulations :
   (a) `break main`
   (b) `run`
   (c) `disassemble`
   (help une commande donne un court mode d'emploi)
3. La visualisation du contenu de la mémoire se fait avec la commande "x" : exemple `(gdb)x /20xb 0x80483c0` permet d'afficher à partir de l'adresse `0x80483c0` les 20 octets (b) suivants en hexadécimal (x) (format `/20xb`)
   (a) Afficher le code mis en mémoire (il faut donc repérer l'adresse de la première instruction et calculer la taille du code)
   (b) Afficher le segment de données.
4. Le contenu des registres peut être affiché en utilisant la commande `info reg` suivie des noms des registres, exemple `info reg eax ebx cs`, sans argument elle affiche tous les registres.
5. L'exécution pas à pas se fait avec la commande `stepi` (`nexti` idem sans rentrer dans les appels de fonctions)
   (a) Avancer pas à pas en affichant à chaque pas le contenu du registre modifié.
   (b) Que se passe-t-il lors de l'exécution de la commande `int $0x80` ?

3 Un petit exercice

Voici quelques informations :

`.skip size , fill` réserve `size` octets contenant `fill` (sinon des 0)

Les numéros correspondant aux appels systèmes sont dans `/usr/include/asm/unistd.h`

Le mode d'emploi est donné par `man` (`man 2 read`), il y a correspondance avec les noms de registres (le numéro de l'appel est dans `eax` ainsi que le retour après l'exécution, ensuite les arguments sont dans l'ordre dans `ebx ecx edx esi edi`)

```
read(int fd, void *buf, size_t count);
eax      ebx        ecx           edx
```

Les descripteurs de fichiers sont dans `/usr/include/unistd.h`

```
/* Standard file descriptors.  */
#define STDIN_FILENO    0       /* Standard input.  */
#define STDOUT_FILENO   1       /* Standard output.  */
#define STDERR_FILENO   2       /* Standard error output.  */
```

Ecrire un programme assembleur qui lit une chaîne de caractère (entrée avec un retour-à-la-ligne), la place ainsi que sa taille dans une partie réservée en mémoire de la section des données, puis l'affiche à l'écran.

CORRECTION du 2

```
JCB@taxus/TESTS 82 % gdb hello
(gdb) break main
Breakpoint 1 at 0x80483c0
(gdb) r
Starting program: /auto/bajard/ENSEIGNEMENT/ARCHI/2003/TESTS/base1
(no debugging symbols found)...(no debugging symbols found)...
Breakpoint 1, 0x080483c0 in main ()
(gdb) disassemble
Dump of assembler code for function main:
0x80483c0 <main>:       mov    $0x4,%eax
0x80483c5 <main+5>:     mov    $0x1,%ebx
0x80483ca <main+10>:    lea    0x8049444,%ecx
0x80483d0 <main+16>:    mov    0x8049453,%edx
0x80483d6 <main+22>:    int    $0x80
0x80483d8 <main+24>:    xor    %eax,%eax
0x80483da <main+26>:    inc    %eax
0x80483db <main+27>:    xor    %ebx,%ebx
0x80483dd <main+29>:    int    $0x80
End of assembler dump.
(gdb) x /31xb 0x80483c0
0x80483c0 <main>:     0xb8    0x04    0x00    0x00    0x00    0xbb    0x01    0x00
0x80483c8 <main+8>:   0x00    0x00    0x8d    0x0d    0x44    0x94    0x04    0x08
0x80483d0 <main+16>:  0x8b    0x15    0x53    0x94    0x04    0x08    0xcd    0x80
0x80483d8 <main+24>:  0x31    0xc0    0x40    0x31    0xdb    0xcd    0x80
(gdb) x /31cb 0x8049444
0x8049444 <data>:   104 'h' 101 'e' 108 'l' 108 'l' 111 'o' 32 ' '  32 ' '  119 'w'
0x804944c <data+8>: 111 'o' 114 'r' 108 'l' 100 'd' 33 '!'  10 '\n' 0 '\0'  15 '\017'
0x8049454 <taille+1>: 0 '\0'  0 '\0'  0 '\0'  65 'A'
(gdb) stepi
0x080483c5 in main ()
(gdb) info reg eax ebx ecx edx
eax            0x4      4
ebx            0x4013be48       1075035720
ecx            0x1      1
edx            0x80483c0        134513600
.....
(gdb) stepi
0x080483d6 in main ()
(gdb) info reg eax ebx ecx edx
eax            0x4      4
ebx            0x1      1
ecx            0x8049444        134517828
edx            0xf      15
(gdb) stepi
hello  world!
0x080483da in main ()
```

CORRECTION 3

```
.data
bonjour:
        .skip  80,0
taille:
        .long

.align 4,'A'

.text

.globl main

main:
      ## display string using read () system call
        movl $3, %eax           # read () system call
        movl $0,%ebx            # %ebx = 0, fd = stdin
        leal bonjour, %ecx      # %ecx ---> hello
        movl $80, %edx       # %edx = count
        int $0x80               # execute read () system call
movl %eax,taille          #retour taille entree


       ## display string using write () system call
        movl $4, %eax           # write () system call
        movl $1,%ebx            # %ebx = 1, fd = stdout
        leal bonjour, %ecx      # %ecx ---> hello
        movl taille, %edx       # %edx = count
        int $0x80               # execute write () system call

        ## terminate program via _exit () system call
        xorl %eax, %eax         # %eax = 0
        incl %eax               # %eax = 1 system call _exit ()
        xorl %ebx, %ebx         # %ebx = 0 normal program return code
        int $0x80               # execute system call _exit ()
```