Université Gustave Eiffel

ESIEE PARIS

Architecture des ordinateurs
TP 2 - Mémoire centrale

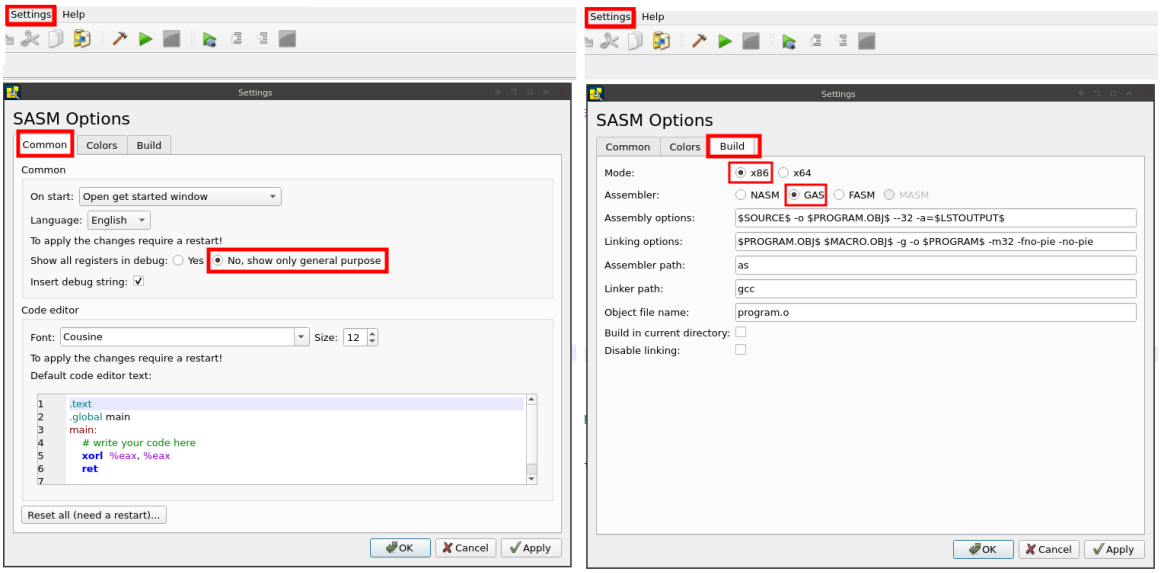
Halim Djerroud

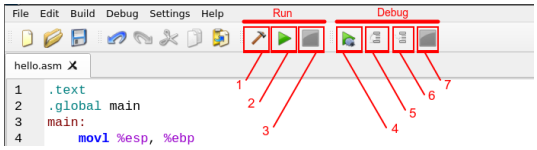
révision 0.1

Mise en route

Si ce n'est pas déjà fait :

```
$ sudo apt update
$ sudo apt install build-essential sasm  gcc gcc-doc gdb gdb-doc binutils gcc-multilib
```





1. (Build) Compiler
2. (Build and run)
3. (Stop)
4. (Debug)
5. (Step over)
6. (Step into)
7. (Stop)

Exercice 1

Créer le fichier exo1.s et insérer le code suivant :

```
  .data
str1: .asciz "Bonjour"

  .bss
  .lcomm buffer, 1024

  .text
  .global _start
  .type _start, @function

_start:
  ret
```

Compiler à l'aide des commandes suivantes :

```
$ as  --32  exo1.s -o exo1.o
$ ld  -m elf_i386 exo1.o -o  exo1
```

1. Si vous exécuter le le ficher généré vous aurez une Erreur de segmentation. Expliquer pour quoi ? Expliquez aussi pourquoi votre programme ne contient pas de symbole `main`
2. A quoi correspond la symbole `_start`
3. Lancer la commande `file exo1`. Consulter le manuel et dire a quoi correspond cette commande.
4. Lancer la commande `readelf -h exo1`. Consulter le manuel et dire a quoi correspond cette commande quand elle est lancée avec l'option `-h`
5. Lancer la commande `readelf -S exo1`. Consulter le manuel et dire a quoi correspond cette commande quand elle est lancée avec l'option `-S`
6. Exécuter la commande : `size exo1`. Donnez la taille de chaque section et taille globale, toutes sections confondues.
7. Exécuter la commande `hexdump -C exo1` ou son alias `hd exo1`. Essayez de comprendre les informations affichées à l'écran. Cherchez, une corrélation avec le résultat de la commande `readelf -S exo1`. Que constatez vous ? Pourquoi la section `.text` (code) vide ?
8. Changez la section `_start` par la section `main` et compilez le code à l'aide de `gcc`. Donnez la commande permettant de compiler le code en 32 bits.

Exercice 2

Créez le fichier exo2.s et insérer le code suivant :

```
.data
  .align 2
var_short:       .word  0x0FFF
.align 4
tab_3_int:       .int   10,20,30
  ma_chaine:       .asciz "Bonjour ASM !"
  .align 8
  var_long_long:   .quad  0xff

  .bss
  .lcomm fuffer, 10

  .text
  .global main

  main:
  movl %esp, %ebp #for correct debugging
  # write your code here
  leal var_short, %eax
  leal fuffer, %ebx
  xorl  %eax, %eax
  ret
```

Compiler à l'aide de la commande suivante :

```
gcc -m32 cours_mem_exemple.s  -g -o mon_pro -no-pie
```

1. Exécuter la commande : `size exo2`
2. Donnez la taille de chaque section et taille globale, toutes sections confondues.
3. Lancer `sasm`, puis visualisez le contenu de la mémoire à l'aide des outils de *debug* fourni dans sasm : (voir Figure 1)

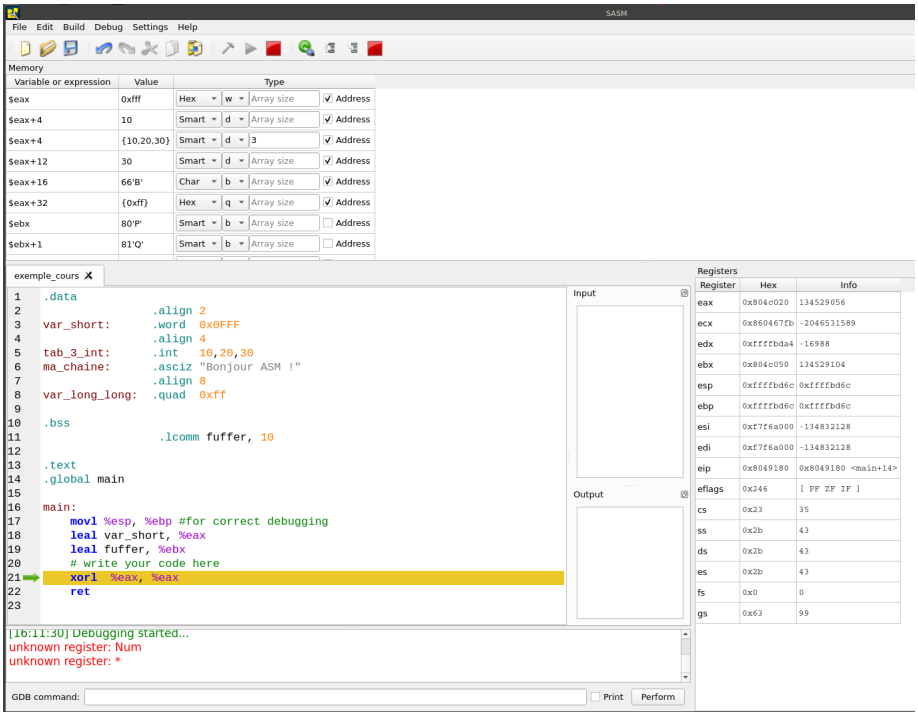


FIGURE 1 – SASM : Consulter le contenu de la mémoire

4. Examiner le contenu de la mémoire à l'aide de l'outil `gdb en ligne de commande` (voir figure 2)

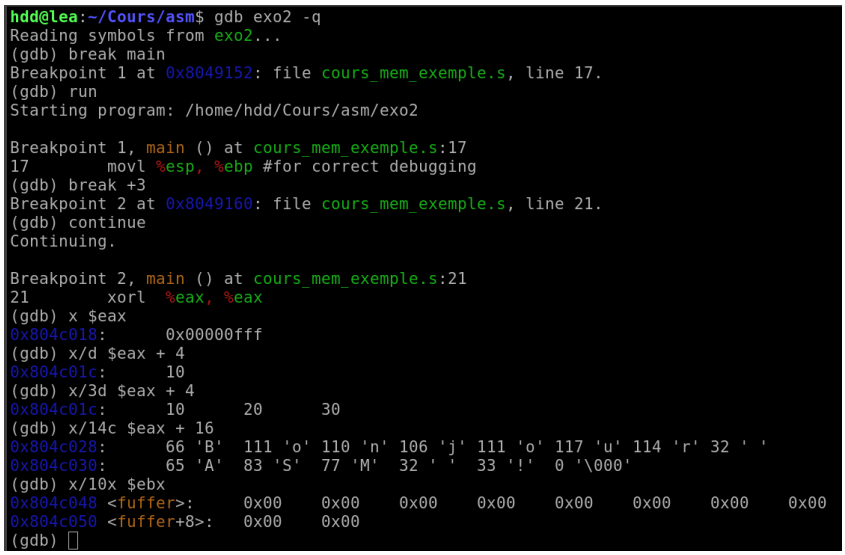


FIGURE 2 – Utilisation de GDB

5. Sur gdb executer la commande suivante : `x/32xw $eax` (voir figure 5)

FIGURE 3 – Afficher plusieurs cases mémoires

Exercice 3 :

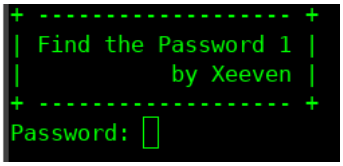
1. Écrire un programme assembleur qui déclare les variables suivantes :

```
char tab[] = {25,12,14};
int x=5;
char c = 'B';
char *ch = "Hello";
```

Exercice 4 :

Sur le site *crackmes.one*

1. Téléchargez le défi suivant : https://crackmes.one/crackme/632cf67b33c5d4425e2cd501
2. Décompressez l'archive dans un dossier, le mot de passe pour l'archive est : `crackmes.one`
3. Exécutez le fichier `findthepassword1` :



4. En utilisant les outils vus dans ce cours, trouvez le mot de passe !