

Architecture des ordinateurs

TP 2 - Mémoire centrale

Halim Djerroud

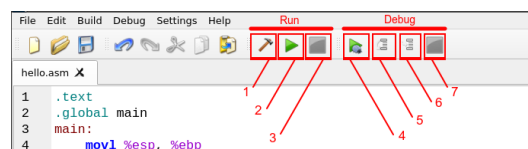
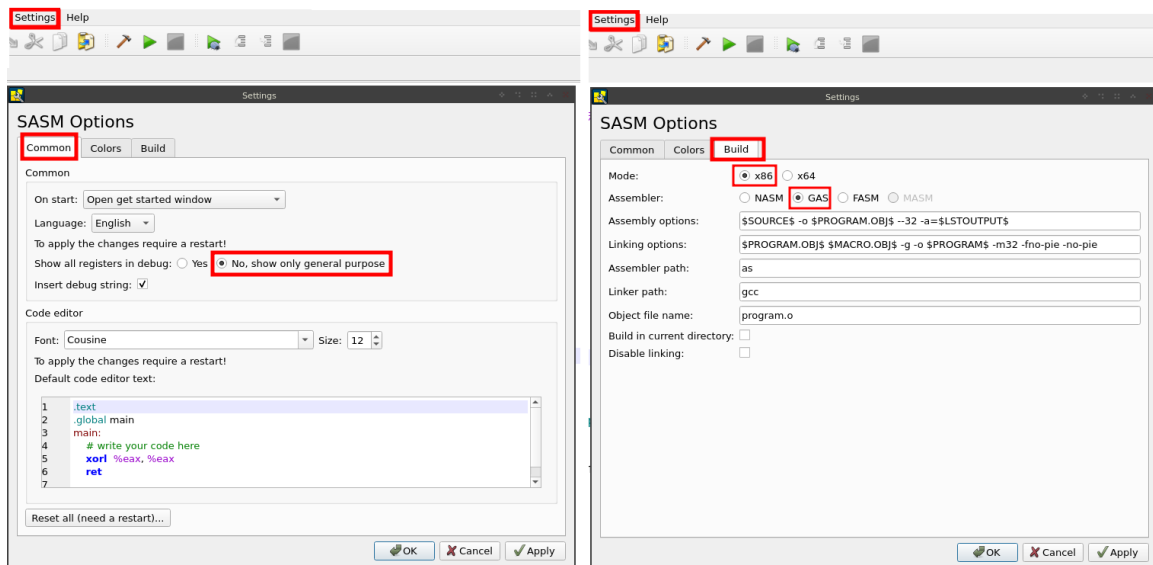
révision 0.1

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 exécutez le fichier 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 le symbole `_start`
3. Lancer la commande `file exo1`. Consulter le manuel et dire à quoi correspond cette commande.
4. Lancer la commande `readelf -h exo1`. Consulter le manuel et dire à quoi correspond cette commande quand elle est lancée avec l'option `-h`
5. Lancer la commande `readelf -S exo1`. Consulter le manuel et dire à 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 insérez 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

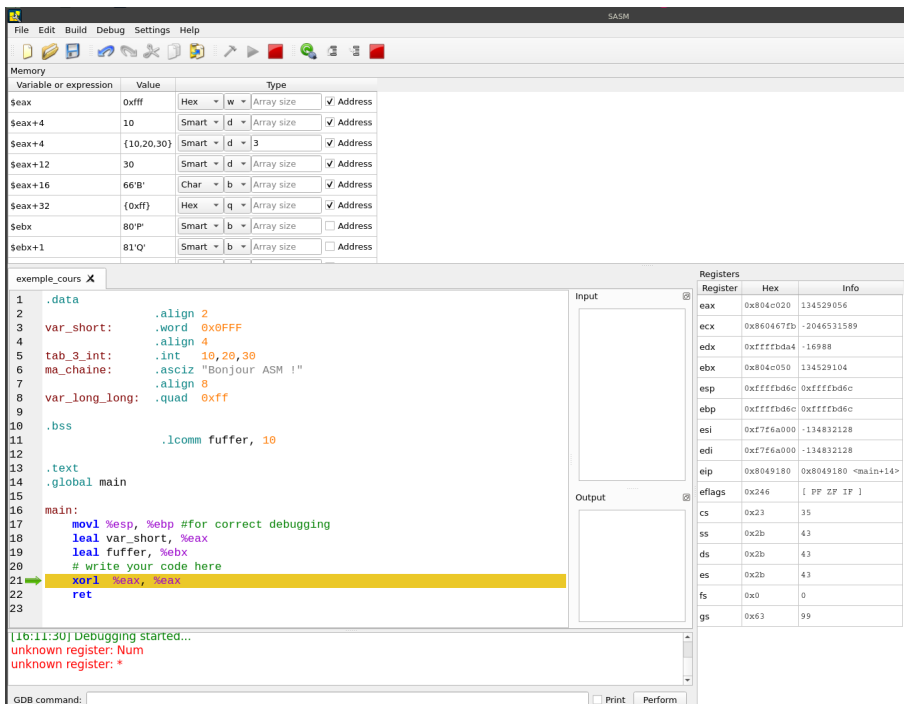
.text
.global main

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)



The screenshot shows the SASM debugger interface. The main window displays the assembly code from the previous block, with line 21 highlighted. On the right, there are two panels: 'Memory' and 'Registers'.

Memory Panel:

Variable or expression	Value	Type
\$eax	0xff	Hex w Array size Address
\$eax+4	10	Smart d Array size Address
\$eax+4	{10,20,30}	Smart d 3 Address
\$eax+12	30	Smart d Array size Address
\$eax+16	66'	Char b Array size Address
\$eax+32	{0xff}	Hex q Array size Address
\$ebx	80'P	Smart b Array size Address
\$ebx+1	81'Q	Smart b Array size Address

Registers Panel:

Register	Hex	Info
eax	0x804c020	134529056
ecx	0x804647fb	-2046531589
edx	0xfffffbd4	-16988
ebx	0x804c050	134529104
esp	0xfffffb6c	0xfffffb6c
ebp	0xfffffb6c	0xfffffb6c
esi	0xf7f6a000	-134832128
edi	0xf7f6a000	-134832128
eip	0x8049180	0x8049180 <main+14>
eflags	0x246	[PF ZF IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	0x63	99

The bottom status bar shows the GDB command: `Print Perform`.

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)

```

hdd@lea:~/Cours/asm$ gdb exo2 -q
Reading symbols from exo2...
(gdb) break main
Breakpoint 1 at 0x8049152: file cours_mem_exemple.s, line 17.
(gdb) run
Starting program: /home/hdd/Cours/asm/exo2

Breakpoint 1, main () at cours_mem_exemple.s:17
17      movl %esp, %ebp #for correct debugging
(gdb) break +3
Breakpoint 2 at 0x8049160: file cours_mem_exemple.s, line 21.
(gdb) continue
Continuing.

Breakpoint 2, main () at cours_mem_exemple.s:21
21      xorl %eax, %eax
(gdb) x $eax
0x804c018: 0x00000fff
(gdb) x/d $eax + 4
0x804c01c: 10
(gdb) x/3d $eax + 4
0x804c01c: 10      20      30
(gdb) x/14c $eax + 16
0x804c028: 66 'B' 111 'o' 110 'n' 106 'j' 111 'o' 117 'u' 114 'r' 32 ' '
0x804c030: 65 'A' 83 'S' 77 'M' 32 ' ' 33 '!' 0 '\000'
(gdb) x/10x $ebx
0x804c048 <fuffer>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x804c050 <fuffer+8>: 0x00 0x00
(gdb)

```

FIGURE 2 – Utilisation de GDB

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

```

(gdb) x/32xw $eax
0x804c018: 0x00000fff 0x0000000a 0x00000014 0x0000001e
0x804c028: 0x6a6e6f42 0x2072756f 0x204d5341 0x00000021
0x804c038: 0x000000ff 0x00000000 0x00000000 0x00000000
0x804c048 <fuffer>: 0x00000000 0x00000000 0x00000000 0x00000000
0x804c058: 0x00000000 0x00000000 0x00000000 0x00000000
0x804c068: 0x00000000 0x00000000 0x00000000 0x00000000
0x804c078: 0x00000000 0x00000000 0x00000000 0x00000000
0x804c088: 0x00000000 0x00000000 0x00000000 0x00000000
(gdb)

```

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 :

```

+ ----- +
| Find the Password 1 |
|           by Xeeven |
+ ----- +
Password: 

```

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