

Avant la séance de travaux pratiques suivante, vous enverrez une archive contenant les fichiers sources ainsi qu'un rapport au format pdf à votre chargé de td.

Arnaud Carayol : arnaud.carayol@univ-mlv.fr
Laurent Braud : laurent.braud@univ-mlv.fr

Vous donnerez à votre mail le sujet suivant:

[Archi L3] [TP2] Nom1--Nom2

Le rapport doit répondre aux questions posées dans le sujet et peut éventuellement contenir des portions de code pour illustrer votre propos. Votre code doit être commenté sinon il ne sera pas lu. Si un fichier que vous rendez n'a pas le comportement attendu, cela doit être dit dans le rapport.

Dans la séance précédente, nous avons vu comment écrire un programme simple entièrement en assembleur. En particulier, nous avons vu comment utiliser les appels système.

À partir de cette séance, nous n'écrirons plus nos programmes intégralement en assembleur. Nous écrirons uniquement une fonction en assembleur et cette fonction sera appelée par un programme C.

Téléchargez les fichiers `add.asm`, `asm_io.inc`, `asm_io.o` et `driver.c`. Le fichier `add.asm` contient le code assembleur d'une fonction nommée `asm_main`. Le fichier `driver.c` est un programme C qui a pour seul but d'appeler la fonction `asm_main`.

Regardons le fichier `add.asm`.

1. La première ligne permet d'avoir accès à des fonctions d'entrée-sortie plus faciles à utiliser que les appels système.

```
%include "asm_io.inc"
```

2. Le code commence et termine par des opérations ayant pour but de respecter la convention d'appel du C. Le sens précis sera vu dans une autre séance.

```
enter    0,0                ; mise en place
pusha
        .
        .
        .
popa
mov      eax, 0              ; retour vers le C
leave
ret
```

3. Cette fonction fait des appels à des fonctions comme `print_nl`, `print_string` et `print_int`. Le code de ces fonctions se trouve dans `asm_io.o`.

- `call print_nl` affiche un retour à la ligne

Question 6 Écrire un programme qui affiche le nombre de bits de `eax` qui valent 1. Par exemple si `eax` vaut `0x0000F00F`, le programme renverra 8.

L'idée est dans un premier temps d'utiliser l'instruction `shr` vue à la question précédente.

Question 7 On cherche une autre manière de réaliser le programme de la question précédente. En traitant quelque exemple, décrivez ce que l'on obtient si l'on réalise le ET logique entre `eax` et `eax-1`. En déduire un autre programme assembleur calculant le nombre de bits valant 1 dans `eax`.

Quel est l'avantage de cette approche ?

Question 8 Récupérez le programme `jdst.asm`. Décrivez la fonction de ce programme et expliquez son fonctionnement. Ne décrivez pas ligne par ligne mais identifiez des blocs et donnez leur sens.