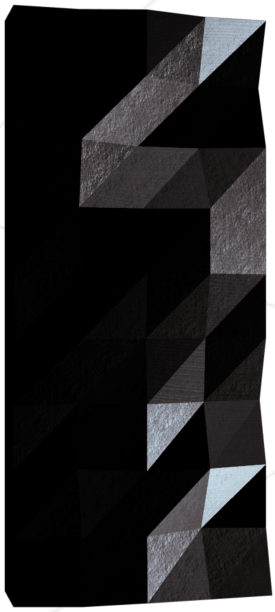

Libasm

Assembly yourself !

Résumé: L'objectif de ce projet est de devnir familier avec le langage assembleur.

Table des matières

Chapitre I

Introduction

Le langage assembleur, souvent appellé asm, est un langage de bas niveau pour ordinateur - ou autre machine programmable - qui possède une très forte correspondance entre le langage et l'architecture de la machine.
Chaque asm est spécifique à un ordinateur particulier. A l'opposé, les langages haut-niveau sont généralement portables sur de multiples plateformes mais nécessitent compilation ou interprétation. Le langage assembleur peut également être appellé langage machine symbolique.

Chapitre II

Instructions générales

- Vos fonctions ne doivent pas quitter de manière innattendue (segmentation fault, bus error, double free, etc) mis à part les comportements indéfinis. Si cela arrive, votre projet sera considéré invalide et vous recevrez un `0` pendant l'évaluation
- Votre `Makefile` doit au minimum contenir les règles `$(NAME)`, `all`, `clean`, `fclean` et `re`. Votre Makefile ne doit compiler/linker que les fichiers nécessaires.
- Pour rendre des bonus dans votre projet, vous devez inclure une règle `bonus` dans votre Makefile, qui ajoutera les différents headers, librairies ou fonctions qui sont interdites dans la partie principale du projet. Les bonus doivent être dans un fichier `_bonus.{c/h}`. L'évaluation de la partie obligatoire est faite séparément de la partie bonus.
- Nous vous encourageons à créer des programmes de test pour vos projets même si ce travail **ne sera ni rendu ni noté**. Cela vous donnera une chance de tester facilement votre rendu et le rendu de vos pairs.
- Rendez votre travail dans le repo git qui vous est assigné. Seul le travail rendu via git sera noté. Si Deepthought doit vous évaluer, cela sera fait après les evaluations de vos pairs. Si une erreur survient pendant l'évaluation de Deepthought, celle-ci s'arrêtera.
- Vous devez écrire en ASM 64 bits. Attention aux "calling convention".
- Vous devez rendre des fichiers '.s' et pas de l'asm inline
- Vous devez compiler votre code avec nasm
- Vous devez utiliser la syntaxe Intel, pas celle d'AT&T.

Chapitre III

Partie obligatoire

- Votre librarie doit s'appeller libasm.a
- Vous devez rendre un main qui testera vos fonctions et compilera avec votre librairie montrer qu'elle fonctionne.
- Vous devez écrire les fonctions suivantes :
  - ft_strlen (man 3 strlen)
  - ft_strcpy (man 3 strcpy)
  - ft_strcmp (man 3 strcmp)
  - ft_write (man 2 write)
  - ft_read (man 2 read)
  - ft_strdup (man 3 strdup, malloc autorisé)
- Vous devez verifier les erreurs lors des syscalls et les renvoyer correctement
- Votre code doit permettre la lecture de la variable errno (de <errno.h>) depuis un fichier .c
- Pour cela, vous avez l'autorisation d'utiliser `extern ___error ou errno_location`

Chapitre IV

Partie bonus

Vous pouvez réecrire les fonctions suivantes en asm. La liste chainée utilisera la structure suivante :

```
typedef struct                 s_list
{
  void                        *data;
  struct s_list       *next;
}                              t_list;
```

- ft_atoi_base (Comme celui de piscine)
- ft_list_push_front (Comme celui de piscine)
- ft_list_size (Comme celui de piscine)
- ft_list_sort (Comme celui de piscine)
- ft_list_remove_if (Comme celui de piscine)