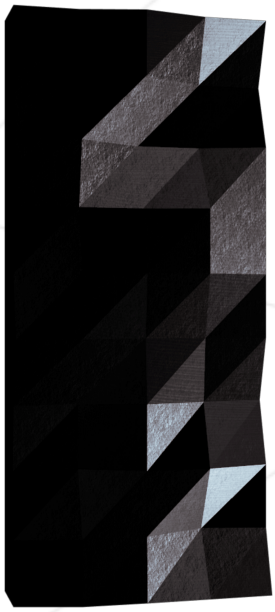

Libftasm

Assembly yourself!

42 staff staff@42.fr

Summary: The aim of this project is to get familiar with assembly language.

Contents

I Foreword **2**

II General Instructions **3**

III Mandatory part **4**

III.1 libftasm introduction 4

III.1.1 Part 1 - Simple libc functions 4

III.1.2 Part 2 - A little less simple libc functions 5

III.1.3 Part 3 - Cat 5

IV Bonus part **6**

Chapter I

Foreword

An assembly (or assembler) language, often abbreviated asm, is a low-level programming language for a computer, or other programmable device, in which there is a very strong (but often not one-to-one) correspondence between the language and the architecture's machine code instructions. Each assembly language is specific to a particular computer architecture. In contrast, most high-level programming languages are generally portable across multiple architectures but require interpreting or compiling. Assembly language may also be called symbolic machine code.

Assembly language is converted into executable machine code by a utility program referred to as an assembler. The conversion process is referred to as assembly, or assembling the source code. Assembly time is the computational step where an assembler is run.

Assembly language uses a mnemonic to represent each low-level machine instruction or opcode, typically also each architectural register, flag, etc. Many operations require one or more operands in order to form a complete instruction and most assemblers can take expressions of numbers and named constants as well as registers and labels as operands, freeing the programmer from tedious repetitive calculations. Depending on the architecture, these elements may also be combined for specific instructions or addressing modes using offsets or other data as well as fixed addresses. Many assemblers offer additional mechanisms to facilitate program development, to control the assembly process, and to aid debugging.

Wikipedia

Chapter II

General Instructions

- This project will be corrected by humans only. You're allowed to organise and name your files as you see fit, but you must follow the following rules.
- The library must be called `libfts.a`.
- You must compile your assembly code with `nasm`.
- You must write 64 bits ASM. Beware of the "calling convention".
- You can't do inline ASM, you must do '.s' files.
- You must use the `Intel` syntax, not the `AT&T`.
- Your `Makefile` must compile the project and must contain the usual rules. It must recompile and re-link the program only if necessary.
- You must submit a `main` that will test your functions and that will compile with your library to show that it's functional.
- Once validated you'll be allowed to use your ASM functions instead of your `C` functions in your `libft` if you feel like it.

```
/* install nasm 2.11.06 */
curl -O http://www.nasm.us/pub/nasm/releasebuilds/2.11.06/macosx/nasm-2.11.06-macosx.zip
uc
mkdir ~/bin
cd nasm-2.11.06
cp nasm ~/bin
echo "export PATH=\$HOME/bin:\$PATH" >> ~/.zshrc
source ~/.zshrc
```

Chapter III

Mandatory part

III.1 libftasm introduction

The `libasm` project's goal is to make you code a small library in ASM. You have to recode some of the basic libc function to be able to generate a library. At the end of this project you should be familiar with the language's syntax, the stack's operating as well as the compilator's behavior.

III.1.1 Part 1 - Simple libc functions

In this first part, you have to recode a set of `libc` functions as described in their respective `man`. Your functions must have the same prototypes and the same behavior as the originals. Their name must be prefixef with `"ft_"`. For example `strlen` becomes `ft_strlen`.

You must recode the following functions:

- `bzero`
- `strcat`
- `isalpha`
- `isdigit`
- `isalnum`
- `isascii`
- `isprint`
- `toupper`
- `tolower`
- `puts` (obviously, you can use the `write` syscall)

III.1.2 Part 2 - A little less simple libc functions

In this part, you must recode some another set of libc functions but with the Instruction Repeat String Operations.

A little bit of documentation

You must recode the following functions:

- `strlen`
- `memset`
- `memcpy`
- `strdup` (obviously you are allowed to call `malloc`)

III.1.3 Part 3 - Cat

To conclude, you must code a `ft_cat` function which will take a `file descriptor` (for example `0`...) as argument and that will have the same behavior as the cat command it'll return void.

`Be careful the context change between the user-space the kernel-space is costly in terms of performance you'll be penalized if you abuse it.`

Chapter IV

Bonus part

We will look at your bonuses if and only if your mandatory part is EXCELLENT. This means that your must complete the mandatory part, beginning to end, and your error management must be flawless, even in cases of twisted or bad usage. If that's not the case, your bonuses will be totally IGNORED.

For the bonus part you are free to add any other function (from the `libc` or not) to your `libftasm` that you want.