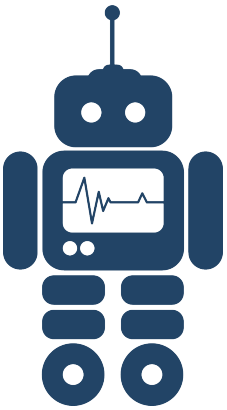
B4 - x86_64 Assembler

B-ASM-400

Bootstrap

In the beginning there was light



1.2

Bootstrap

language: x86-64 Assembly

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

You will use the following tools:

- compiler: `nasm -f elf64 file.asm`
- linker: `ld -o executable *.o`
- filename extensions: .asm for Assembly source files, .inc for includes
- ABI & calling conventions: System V AMD64

RTFM
The Instruction Reference Manual may interest you.
The Internet, as a whole, is an obviously extensive source of information – given the right keywords.

`gdb` may prove very useful. Here are some simple commands that could help you:

- switch to the Intel syntax: `set disassembly-flavor intel`
- disassemble a function/label: `disassemble label_name`
- set a breakpoint: `b *0x7fffffffdeadc0de`
- display memory contents: `x/x $rbx` (the address being stored in the RBX register)
- display a string: `x/s $rbp` (the string address being stored in the RBP register)
- next instruction (after a breakpoint): `ni`
- continue execution (after a breakpoint): `continue`

Pure Assembly

Hello, World!

In assembly, write two versions of a program that writes `Hello, World!` on the standard output:

- one using the `write` **system call**;
- one using the `printf` **function**.

Don't forget to call the `exit` function/system call or to return a correct value at the end of your main function.

You may take some inspiration from kick-off slideshow's example code.

If you do reproduce the aforementioned example code, you will realize that something is wrong with it and that it will most likely crash during its execution.
Hint: this has something to do with function calling conventions.

String Length

In Assembly, write a `my_strlen` function that behaves exactly like the standard `strlen` function. Then test it with a `main` function implemented in Assembly too.

`man strlen` if you can't recall. But, really, you should remember by now.

Mixing Assembly with C

Direct Linking

In Assembly, write a function called `disp_string` that takes one string argument and displays it on the standard output, followed by its length (cf. the example below for formatting reference). Use the `printf` function from the C standard library for the output and your `my_strlen` function.

Then write a `main` function in C that calls `disp_string` for every argument given to the program.

Compile and link the two parts into a single executable named `disp_args`.

```
Terminal
~/B-ASM-400> ./disp_args Epitech ASM rulez
Epitech: 7
ASM: 3
rulez: 5
```

Static Library

Create a library named `libds.a` containing the `disp_string` you wrote earlier. Use it with your `main` function, which was written in C.

Nothing really new here.

Dynamic Library

Do the same thing except that this time, the library is dynamic: `libds.so`. If you take the same code as before, you should have a compilation/linking problem.

This is due to your code not being position-independent, which is required for x86-64 libraries for memory usage and performance reasons. Your job here is to make your code conformant.

About position-independent code (PIC): the Internet is your friend.
About writing PIC code in `nasm`: here.