



SE 2XA3 (2019/20, Term I) Minor Lab 4 -- lab section L03

[Back To Lab Menu](#)
[Back To Main Menu](#)
[Log Out](#)

If you are in the correct lab room and during the SCHEDULED time slot for your lab section for Minor Lab 4 and working on one of the lab's workstations, go into Marks+Comments to see if your attendance was registered. If you signed on the course website too early (the most common problem), your attendance would not be recorded; in such case log out and sign on again. If your attendance is still not recorded, ask your TA to record your attendance and email Prof. Franek right away. If you are not using one of the lab's workstations and want to have your attendance recorded, tell the TA.

The Minor Labs are open book labs, you can bring and use any notes etc., you can access any website, Google, Wikipedia etc. You are allowed to cooperate with other people, inside or outside the lab. The TA can provide extensive individual help as long as this would not deprive other students of the TA's help.

Before you start working on any of the programs in the lab project,

1. download [driver.c](#) to your workstation / laptop, transfer it to **moore**, and modify it to a UNIX text file by `dos2unix`;
2. download [simple_io.inc](#) to your workstation / laptop, transfer it to **moore**, and modify it to a UNIX text file by `dos2unix`;
3. download [simple_io.asm](#) to your workstation / laptop, transfer it to **moore**, and modify it to a UNIX text file by `dos2unix`;
4. download [makefile](#) to your workstation / laptop, transfer it to **moore**, and modify it to a UNIX text file by `dos2unix` and tailor it for your purpose in this lab project, you may instead prefer to create your own makefile from scratch, or to download the makefile from one of the sample solutions below.

Task 1. Write NASM program lab4_1.asm

The program has an integer variable (hence of size `qword`), say named `x` (you can call it whatever you want) initialized to a value of 4. It computes the square of the value of `x` and displays it using the `simple_io` subroutines. The program should display exactly (including the spaces) `4 * 4 = 16`

You will need the NASM instruction for multiplication of unsigned integers. You may use `MUL source` which computes `RAX*source` and stores the result in `RAX`. The `source` can be memory or a register. If it is a memory, you must specify the size, i.e. `qword`. The best approach is to have a look at [first.asm](#) for inspiration.

When assembled and linked, executing `lab4_1` program should display

```
4 * 4 = 16
```

If you change the value of `x` from 4 to 5, re-assemble and re-link the program, and execute it, you should see

```
5 * 5 = 25
```

a sample solution for [lab4_1.asm](#), and the necessary [makefile](#)

Task 2. Write NASM program lab4_2.asm

The program has an integer variable (hence of size `qword`), say named `x` (you can call it whatever you want) initialized to a value of 5. It computes the value of the polynomial $x^2 + 3x - 5$ and displays it using `simple_io` subroutines. The program should display exactly (including the spaces -- this time there are no spaces around `*`)

```
5*5 + 3*5 - 5 = 35.
```

You will need NASM instruction for addition (use `ADD` as we discussed it in class), and multiplication of unsigned integers;

for that you may again use `MUL source`, see Task 1. Note, this is just a fancier version of `lab4_1.asm` from Task 1.

When assembled and linked, executing `lab4_2` program should display

$5 * 5 + 3 * 5 - 5 = 35$

If you change the value of `x` from 5 to 7, re-assemble and re-link the program and execute it, you should see

$7 * 7 + 3 * 7 - 5 = 65$

a sample solution for `lab4_2.asm`, and the necessary `makefile`

Task 3. Write NASM program `lab4_3.asm`

Make a program that has in the `.data` section a C string `"123456"`, two `qwords` initialized to integer values of 1 and 2, respectively, and a byte initialized to a character `'A'`. The program displays the string on a single line using `print_string` and `print_nl` subroutines, and then it displays the two stored integers (with the values 1 and 2) on a line separated by a bunch of spaces using `print_int`, `print_char` and `print_nl`, and then on the next line it displays the value of the stored character, i.e. `A`, using `print_char` and `print_nl`. The program then rewrites the value of the stored string to `"ABC"` (it again must be a C string terminated by null), the value of the stored integers to 7 and 8, and the value of the stored character to `'B'` and repeats the printing.

a sample solution for `lab4_3.asm`, and the necessary `makefile`

Task 4. Write NASM program `lab4_4.asm`

The detailed requirements are in the skeleton of the program `lab4_4-skel.asm` in the form of comments. Here we give an overview of the program.

- The program expects two command line arguments (the name of the program and one argument).
- It checks the number of command line arguments and if it is not 2, the program terminates with an error message.
- Then the program checks the first command line argument (i.e. `argv[1]`) whether it is a string `"1"` or `"2"`. If neither, the program terminates with an error message. *The way to do it is to check the first byte of `argv[1]` and it should be a char 1 or a char 2, and then the second byte of the argument and it should be 0 (null).*
- If the command line argument was `"1"`, the `display_array` subroutine is called with (integer) parameter 1, otherwise with (integer) parameter 2.
- The subroutine `display_array` displays either the array `arr1` (if the parameter was 1) or the array `arr2` (if the parameter was 2).
- The subroutine `display_array` displays the array by traversing it left to right and displaying the content of each item encountered, separated by commas (no comma after the last item).

Sample runs:

```
[teststudent@moore] lab4_4
incorrect number of command line arguments
[teststudent@moore] lab4_4 a b
incorrect number of command line arguments
[teststudent@moore] lab4_4 a
incorrect command line argument
[teststudent@moore] lab4_1 1
1,2,3,4,5,6,7,8,9,10
[teststudent@moore] lab4_1 2
11,12,13,14,15,16,17,18,19,20
```

a sample solution for `lab4_4.asm`, and the necessary `makefile`

Task 5. Write NASM program `lab4_5.asm`

The detailed requirements are in the skeleton of the program `lab4_5-skel.asm` in the form of comments. Here we give an overview of the program.

- Your task is to program the subroutine `line1`.
- The subroutine expects one parameter, an address `a` where an integer `N1` is stored. It finds `N2` at address `a+36`, and `N3` at address `a+72`.

- The `asm_main` subroutine sets it up properly and makes a proper call to `line1`, so you do not have to worry about it. Your task is solely to program `line1` subroutine.
- What `line1` is supposed to do is explained in the comments in the skeleton, but we describe it here briefly as well.
- `line1` displays a line on the screen that is composed like this:
 $12-N_1$ dots, N_1 pluses, one `|`,
 N_1 pluses, $12-N_1$ dots, followed by the same pattern for N_2 and followed by the same pattern for N_3

For instance, if $N_1=9$, $N_2=9$, $N_3=9$, the display will look like this

```
...+++++++|+++++++.....+++++++|+++++++.....+++++++|+++++++...
```

For instance, if $N_1=4$, $N_2=2$, $N_3=1$, the display will look like this

```
.....++++|++++.....++|++.....+|+.....
```

For instance, if $N_1=3$, $N_2=0$, $N_3=0$, the display will look like this

```
.....+++|+++.....|.....|.....
```

- Hence you need an external loop for N_1 , N_2 , and N_3 , and a whole bunch of internal loops: a loop to store $12-N_1$ dots in `line`, then another loop to store N_2 pluses in `line`, then store `'|'` in `line`, then another loop to store N_2 pluses, and a loop to store $12-N_1$ dots in `line` (note that you can cut and paste the code to speed up the coding as the same loops are repeated).
- After all the loops are done, the `line` is displayed.
- Note, that you can use for testing various values of N_1 , N_2 , N_3 , see `asm_main` subroutine.

a sample solution for [lab4_5.asm](#), and the necessary [makefile](#)