**Oтчёт по лабораторной работе №8**

**Дисциплина: архитектура компьютера**

**Панявкина Ирина Васильевна**

**НКАбд-04-24**

Содержание

[1 Цель работы 3](#__RefHeading___Toc301_744737701)

[2 Задание 3](#__RefHeading___Toc303_744737701)

[3 Теоретическое введение 3](#__RefHeading___Toc305_744737701)

[4 Выполнение лабораторной работы 4](#__RefHeading___Toc307_744737701)

[4.1 Реализация циклов в NASM 4](#__RefHeading___Toc309_744737701)

[4.2 Обработка аргументов командной строки 11](#__RefHeading___Toc317_744737701_Copy_1)

[4.3        Выполнение заданий для самостоятельной работы 15](#__RefHeading___Toc694_1879274325)

[5 Выводы 17](#__RefHeading___Toc321_744737701)

[Список литературы 18](#__RefHeading___Toc323_744737701)

# 1 Цель работы

Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

# 2 Задание

1.       Реализация циклом в NASM

2.       Обработка аргументов командной строки

3.       Самостоятельное написание программы по материалам лабораторной работы

# 3 Теоретическое введение

Стек — это структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды.

Основной функцией стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Кроме того, в нём выделяется память для локальных переменных и могут временно храниться значения регистров.

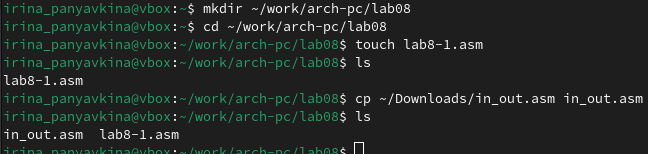
Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается. Для стека существует две основные операции:

* добавление элемента в вершину стека (push);
* извлечение элемента из вершины стека (pop).

# 4 Выполнение лабораторной работы

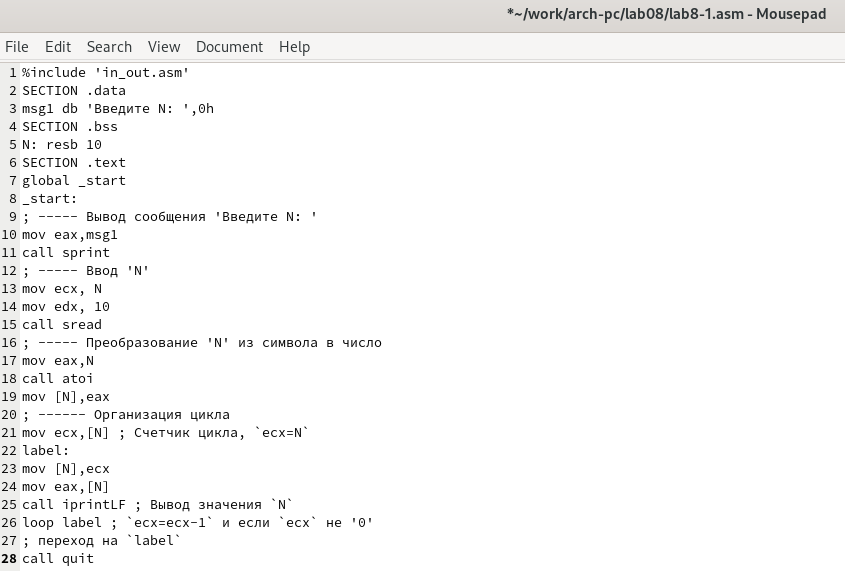
## 4.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы №8, а также файл lab8-1.asm и копирую в текущий каталог файл in\_out.asm с помощью утилиты cp, т.к. он будет использоваться в других программах (рис. 4.1).

**

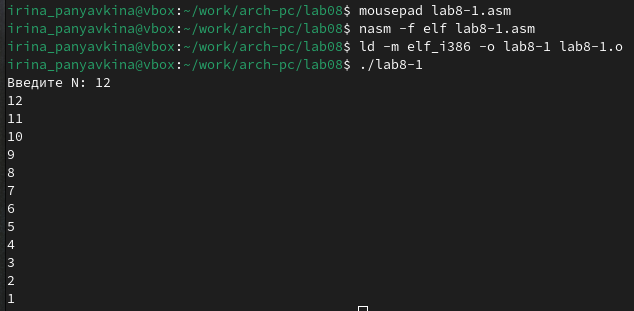
*Рис. 4.1: Создание каталога и файла для программы, создание копии внешнего файла*

Открываю созданный файл и вставляю в него скопированную программу из листинга (рис. 4.2).



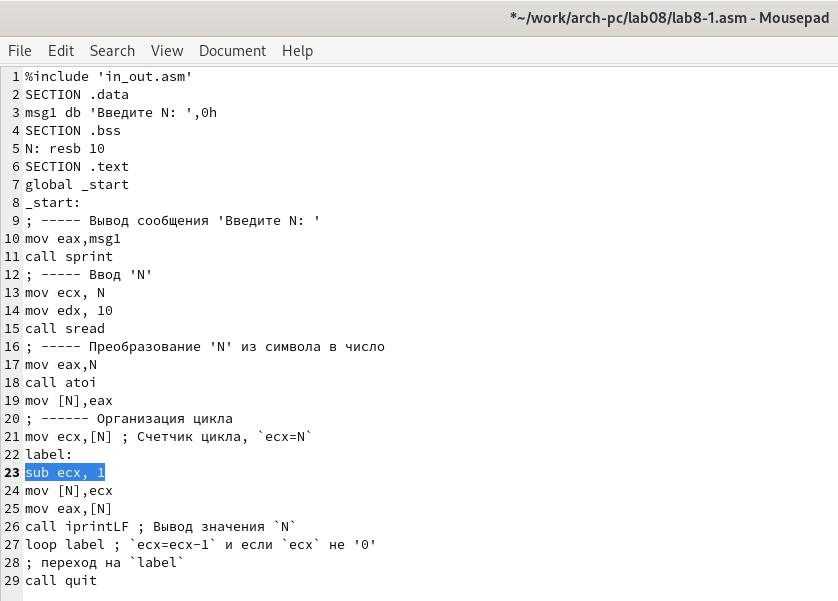
*Рис. 4.2: Редактирование файла и сохранение программы*

Запускаю программу, убеждаюсь в том, что она показывает работу циклов в NASM (рис. 4.3).



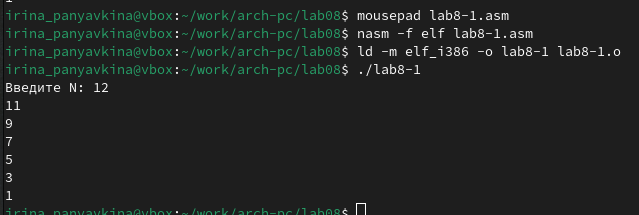
*Рис. 4.3: Запуск исполняемого файла*

Изменяю программу таким образом: в теле цикла я изменяю значение регистра ecx (рис. 4.4).



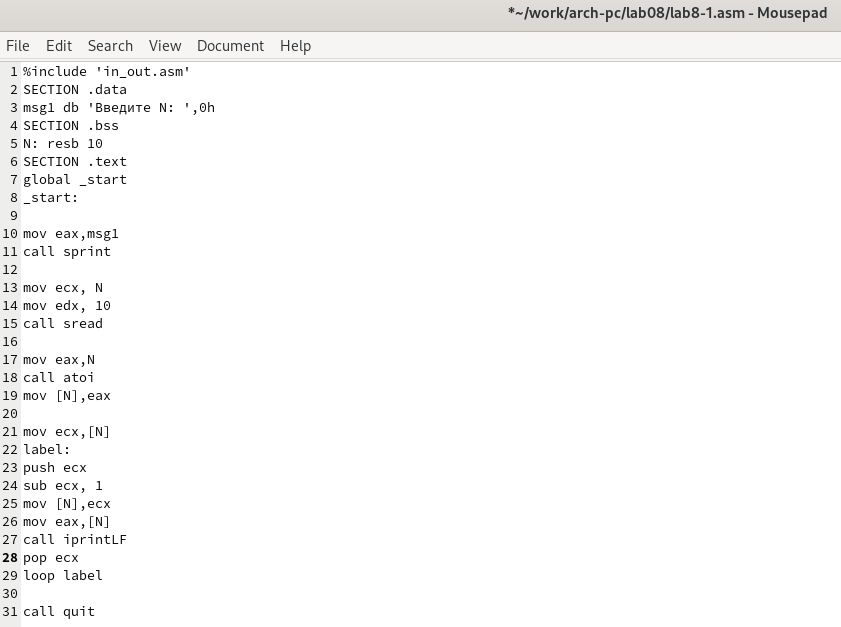
*Рис. 4.4: Редактирование файла*

Теперь из-за того, что регистр ecx на каждой итерации уменьшается на 2 значения, количество итераций так же уменьшается в два раза (рис. 4.5).



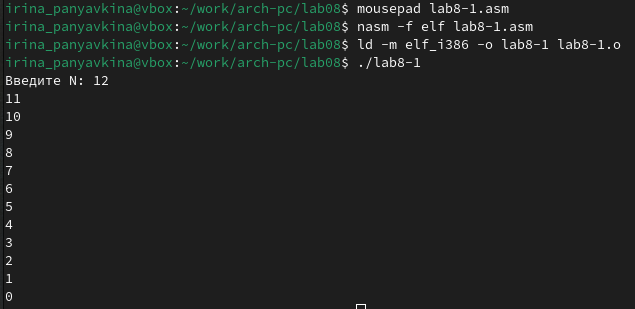
*Рис. 4.5: Запуск исполняемого файла (изменённой программы)*

Редактирую программу, добавляя в неё команды push и pop (рис. 4.6).

**

*Рис. 4.6: Редактирование файла и сохранение программы*

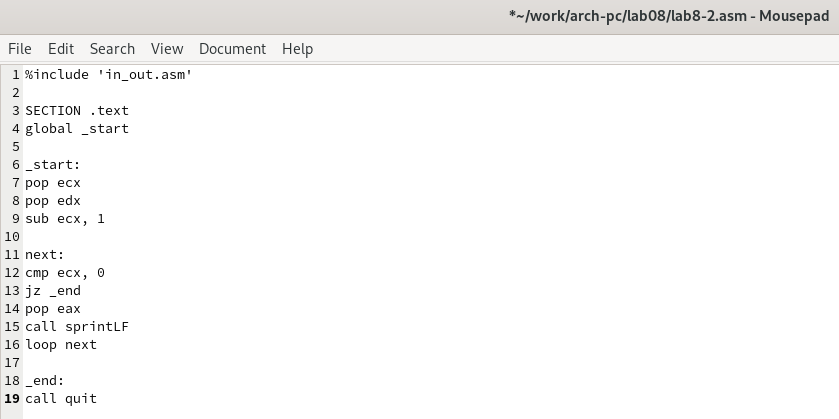
Вижу, что теперь количество итераций совпадает введённому N, однако произошло смещение выводимых чисел на -1 (рис. 4.7).



*Рис. 4.7: Запуск исполняемого файла (изменённой программы)*

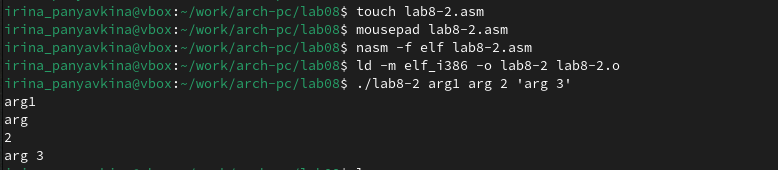
## 4.2 Обработка аргументов командной строки

Создаю новый рабочий файл lab8-2.asm и вставляю в него скопированную программу из следующего листинга (рис. 4.8)



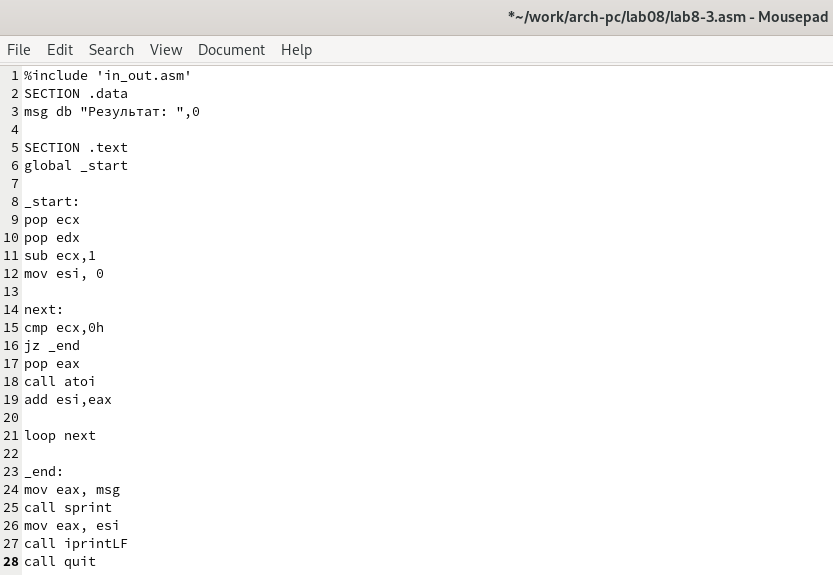
*Рис. 4.8: Сохранение новой программы из листинга*

Компилирую программу и запускаю, указав аргументы. Программа обработала то же количество аргументов, что и вводилось (рис. 4.9).

**

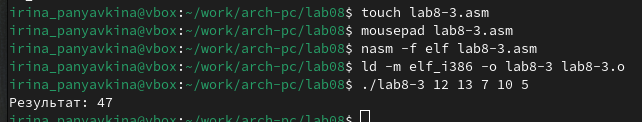
*Рис. 4.9: Компиляция и запуск исполняемого файла с аргументами*

Создаю новый рабочий файл lab8-3.asm и вставляю в него скопированную программу из третьего листинга (рис. 4.10).



*Рис. 4.10: Сохранение новой программы из листинга*

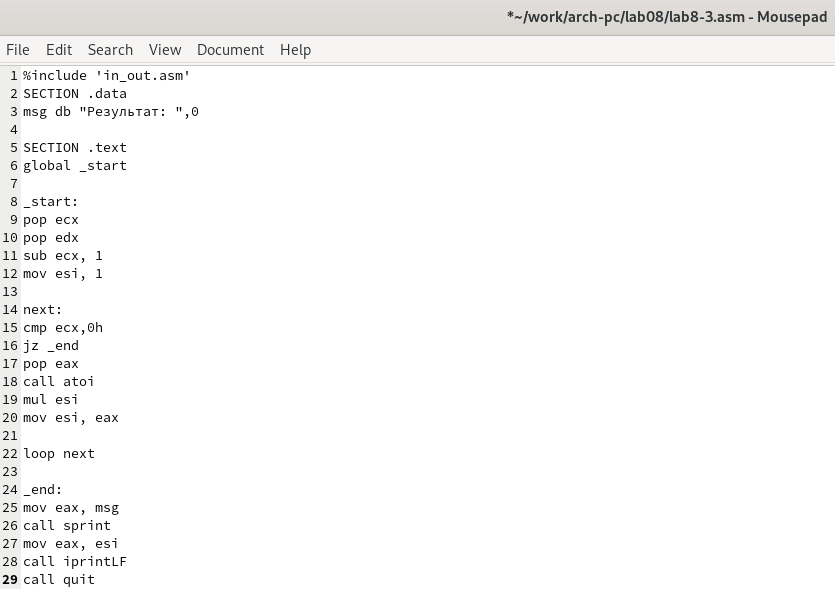
Компилирую программу и запускаю, указав в качестве аргументов некоторые числа, программа должна их складывать (рис. 4.11).



*Рис. 4.11: Компиляция и запуск исполняемого файла с аргументами*

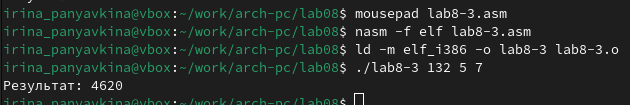
Программа работает корректно, выдаёт верный результат.

Затем изменяю программу так, чтобы указанные аргументы умножались, а не складывались (рис. 4.12).



*Рис. 4.12: Редактирование файла и сохранение программы*

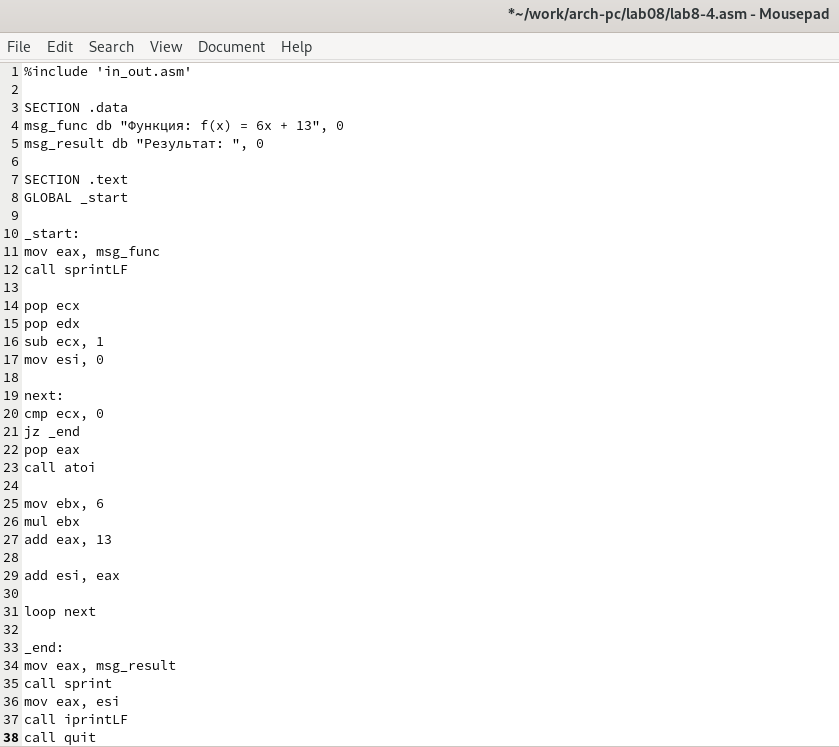
Теперь программа умножает введённые числа, результат верный (рис. 4.13).



*Рис. 4.13: Запуск исполняемого файла (изменённой программы)*

## 4.3        Выполнение заданий для самостоятельной работы

При выполнении 6 лабораторной работы, с помощью программы я выяснила, что мой вариант — 15. Мне нужно написать программу, использовав следующую функцию: f(x)=6x+13 (рис. 4.14).



*Рис. 4.14: Программа самостоятельной работы*

Код первой программы:

%include 'in\_out.asm'

SECTION .data

msg\_func db "Функция: f(x) = 6x + 13", 0

msg\_result db "Результат: ", 0

SECTION .text

GLOBAL \_start

\_start:

mov eax, msg\_func

call sprintLF

pop ecx

pop edx

sub ecx, 1

mov esi, 0

next:

cmp ecx, 0

jz \_end

pop eax

call atoi

mov ebx, 6

mul ebx

add eax, 13

add esi, eax

loop next

\_end:

mov eax, msg\_result

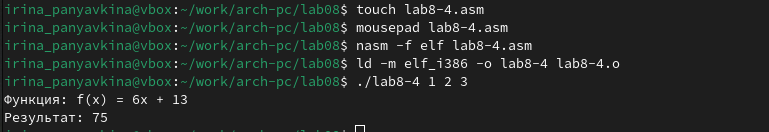
call sprint

mov eax, esi

call iprintLF

call quit

Проверяю корректность написания программы и её работу, указав в качестве аргумента несколько чисел (рис. 4.15).



*Рис. 4.15: Запуск исполняемого файла*

Программа работает успешно, результат получается правильным — соответствует условию задания.

# 5 Выводы

Выполнив данную лабораторную работу, я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки.

# Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.

2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.

3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. Org/.

4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O’Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.

6. Robbins A. Bash Pocket Reference. — O’Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.

7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.

8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.

9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.

10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.

11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.

12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.

13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.

14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: <http://www.stolyarov.info/books/asm_unix>.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).