

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ
Факультет физико-математических и естественных наук
Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №8
дисциплина: Архитектура компьютеров и операционные системы

Студент: Галиев Самир Салаватович

Группа: НКАбд-02-25

МОСКВА

2025 г.

Оглавление

1. Цель работы - стр. 3
2. Результаты выполнения лабораторной работы - стр. 3
 - 2.1 Реализация циклов в NASM - стр. 3
 - 2.2 Обработка аргументов командной строки - стр. 7
 - 2.3 Вычисление суммы и произведения аргументов - стр. 8
3. Задание для самостоятельной работы - стр. 9
4. Выводы - стр. 10
5. Список литературы - стр. 11

1. Цель работы

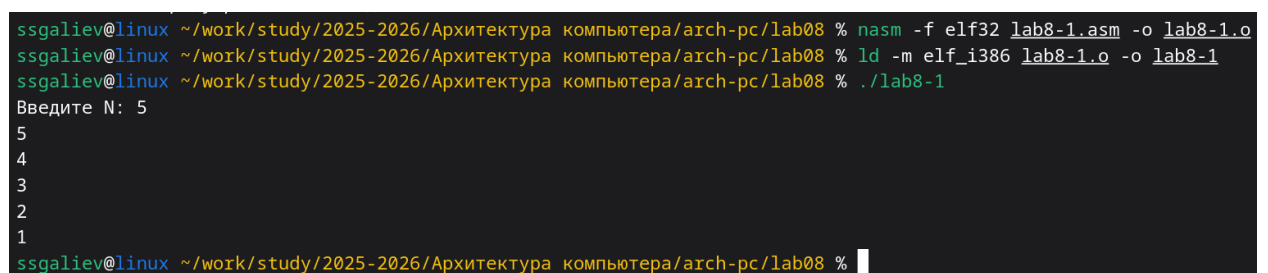
Приобретение навыков написания программ на ассемблере NASM с использованием циклических конструкций (loop) и обработки аргументов командной строки.

2. Результаты выполнения лабораторной работы

2.1. Реализация циклов в NASM

Задание: Написать программу lab8-1.asm, запрашивающую число **N** и выводящую значения от **N** до **1** с помощью loop. Затем изменить тело цикла, добавив sub esx, 1, и проанализировать поведение. После этого исправить программу с использованием стека (push/pop).

Выполнение:



```
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % nasm -f elf32 lab8-1.asm -o lab8-1.o
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % ld -m elf_i386 lab8-1.o -o lab8-1
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % ./lab8-1
Введите N: 5
5
4
3
2
1
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 %
```

Рисунок 1: вывод программы lab8-1.asm

Комментарий: Программа корректно работает: при вводе $N = 5$ выводится последовательность 5, 4, 3, 2, 1. Регистр esx используется только инструкцией loop, что гарантирует точное число итераций.

```
GNU nano 8.7
#include 'in_out.asm'

SECTION .data
msg1 db 'Введите N: ',0h

SECTION .bss
N: resb 10

SECTION .text
global _start
_start:
    mov eax, msg1
    call sprint
    mov ecx, N
    mov edx, 10
    call sread
    mov eax, N
    call atoi
    mov [N], eax
    mov ecx, [N]
label:
    sub ecx, 1
    mov [N], ecx
    mov eax, [N]
    call iprintLF
    loop label

    call quit
```

Рисунок 2: Корректировка кода

Комментарий: Добавляем `sub ecx, 1` над `mov [N], eax`

```
4294841258
4294841256
4294841254
4294841252
4294841250
4294841248
4294841246
4294841244
4294841242
4294841240
4294841238
4294841236
4294841234
4294841232
4294841230
4294841228
4294841226
4294841224
4294841222
4294841220
4294841218
4294841216
4294841214
4294841212
4294841210
4294841208
4294841206
4294841204
4294841202
4294841200
4294841198
4294841196
4294841194
4294841192
4294841190
4294841188
4294841186
4294841184
4294841182
4294841180
4294841178
4294841176
4294841174
4294841172
4294841170
42948411
```

Рисунок 3: *бесконечный цикл* программы

Комментарий: После добавления `sub esx, 1` в тело цикла программа уходит в бесконечный цикл, выводя огромные значения (например, 4294967295, 4294967294 и т.д.). Это происходит потому, что `esx` уменьшается **дважды** (в `sub` и в `loop`), быстро переходит в отрицательные значения, которые интерпретируются как большие беззнаковые числа — условие завершения цикла никогда не достигается.

```
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % nano lab8-1.asm
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % nasm -f elf32 lab8-1.asm -o lab8-1.o
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % ld -m elf_i386 lab8-1.o -o lab8-1
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % ./lab8-1
Введите N: 5
4
3
2
1
0
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 %
```

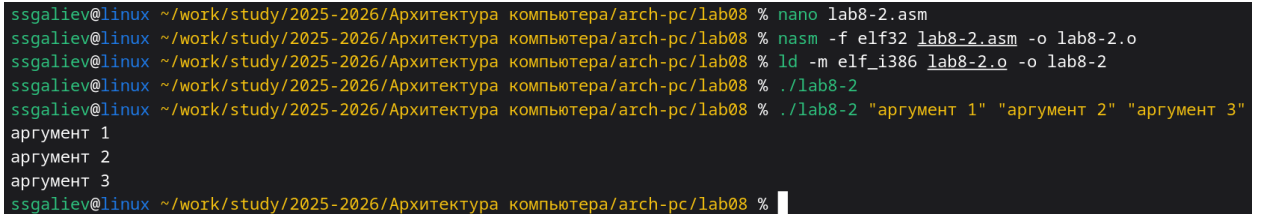
Рисунок 4: Корректный вывод программы(исправление кода)

Комментарий: С использованием `push ecx` перед изменением и `pop ecx`, после значение счётчика сохраняется. Цикл снова выполняется ровно N раз. Вывод корректен: 4, 3, 2, 1, 0.

2.2. Обработка аргументов командной строки

Задание: Написать программу lab8-2.asm, которая извлекает аргументы командной строки из стека и выводит их по одному на отдельной строке.

Выполнение:



```
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % nano lab8-2.asm
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % nasm -f elf32 lab8-2.asm -o lab8-2.o
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % ld -m elf_i386 lab8-2.o -o lab8-2
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % ./lab8-2
аргумент 1
аргумент 2
аргумент 3
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 %
```

Рисунок 5: Запущенная программа с заранее введенными текстовыми значениями

Комментарий: Программа запущена с тремя аргументами: alpha, beta, gamma. Все три аргумента успешно извлечены и выведены. Имя программы (./lab8-2) проигнорировано за счёт sub ecx, 1.

2.3. Вычисление суммы и произведения аргументов

Задание: Реализовать программу lab8-3.asm для вычисления **суммы** числовых аргументов. Затем модифицировать её для вычисления **произведения**.

Выполнение:

```
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % ./lab8-3 10 20 5
Результат:35
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 %
```

Рисунок 6: Программа запущена с аргументами 20 10 5. Выведено: «Результат: 35» — корректная сумма.

```
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % nano lab8-4.asm
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % nasm -f elf32 lab8-4.asm -o lab8-4.o
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % ld -m elf_i386 lab8-4.o -o lab8-4
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % ./lab8-4 2 3 4
Произведение:24
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 %
```

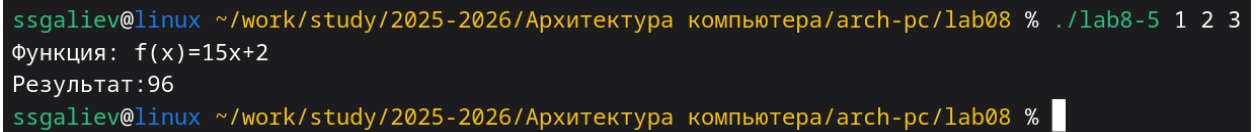
Рисунок 7: Программа запущена с аргументами 2 3 4. Выведено: «Произведение: 24» — корректное произведение. Использована инструкция `mul` и начальное значение `esi = 1`.

Комментарии: Сумма и произведение у обеих программ выведены корректно. Задание выполнено верно.

3. Задание для самостоятельной работы

Задание: Написать программу, вычисляющую сумму значений функции $f(x)$ для всех переданных аргументов x_1, x_2, \dots, x_n . Вид функции выбирается из таблицы 8.1 по варианту (вариант 11)

Выполнение:



```
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 % ./lab8-5 1 2 3
Функция: f(x)=15x+2
Результат:96
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab08 %
```

Рисунок 8: Программа запущена с аргументами 1 2 3.

Комментарии:

Вычисления:

- $f(1)=15 \cdot 1 + 2 = 17$
 - $f(2)=15 \cdot 2 + 2 = 32$
 - $f(3)=15 \cdot 3 + 2 = 47$
- Сумма: $17 + 32 + 47 = 96$

Результат совпадает с ожидаемым — задание выполнено верно.

4. Выводы

- 1) Инструкция `loop` использует регистр `ecx` как беззнаковый счётчик. Любое изменение `ecx` внутри цикла нарушает логику и может привести к бесконечному циклу с выводом мусорных значений.
- 2) Стек (`push/pop`) позволяет временно использовать `ecx` в вычислениях, не нарушая работу цикла.
- 3) Аргументы командной строки передаются через стек в обратном порядке; первыми извлекаются `argc` и имя программы.
- 4) Возможна реализация арифметических функций над аргументами: сумма, произведение, линейные функции вида $ax+b$.

Цель работы полностью достигнута.

5. Список литературы:

- 1) Демидова А. В. *Архитектура ЭВМ. Лабораторная работа №8.*
- 2) Столяров А. *Программирование на языке ассемблера NASM для ОС Unix.* — М.: МАКС Пресс, 2011.
- 3) The NASM documentation. — <https://www.nasm.us/docs.php>
- 4) GNU Assembler Guide. — Arch Linux Wiki.