

Лабораторная работа 8

Программирование цикла. Обработка аргументов командной строки

Головина Мария Игоревна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	10
5	Выводы	23
	Список литературы	24

Список иллюстраций

4.1	Создание каталога и файла для выполнения лабораторной работы	10
4.2	Листинг 1	11
4.3	Результаты работы программы из листинга 1 методического указания	11
4.4	Листинг 1 с внесенными изменениями согласно методическому указанию	12
4.5	Результаты работы программы из Листинга 1 с внесенными изменениями согласно методическому указанию	13
4.6	Измененный Листинг 1 с внесенными командами push и pop . . .	14
4.7	Результаты работы программы измененного Листинга 1 с внесенными командами push и pop	15
4.8	Листинг 2	16
4.9	Результат работы программы из Листинга 2	16
4.10	Листинг 3	17
4.11	Результат работы программы из Листинга 3	18
4.12	Измененный Листинг 3 для вычисления произведения аргументов командной строки	19
4.13	Результаты работы программы для вычисления произведения аргументов командной строки	20
4.14	Листинг самостоятельного задания №1	21
4.15	Результаты работы программы по самостоятельному заданию №1	22

Список таблиц

1 Цель работы

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

2 Задание

1. Создать каталог для программ лабораторной работы № 8, перейти в него и создать файл lab8-1.asm
2. Ввести в файл lab8-1.asm текст программы из листинга 1 методического указания. Создать исполняемый файл и запустить его. Посмотреть результаты работы.
3. Изменить текст программы, добавив изменение значения регистра esx в цикле. Создать исполняемый файл и запустить его. Посмотреть результаты работы. Написать пояснение по результатам работы программы.
4. Внести изменения в текст программы, добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop . Создать исполняемый файл и проверить его работу. Написать пояснение по результатам работы программы.
5. Создать файл lab8-2.asm и ввести в него текст программы из листинга 2 методического указания. Создать исполняемый файл и запустить его, указав необходимые аргументы.
6. Создать файл lab8-3.asm и ввести в него текст программы из листинга 3 методического указания. Создать исполняемый файл и запустить его, указав необходимые аргументы. Написать пояснение по результатам работы программы.

7. Изменить текст Листинга 3 для вычисления произведения аргументов командной строки. Проверить результаты работы программы.

Задание для самостоятельной работы

1. Напишите программу, которая находит сумму значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$, т.е. программа должна выводить значение $f(x_1) + f(x_2) + \dots + f(x_n)$. Значения x_i передаются как аргументы. Вид функции $f(x)$ выбрать из таблицы вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Создайте исполняемый файл и проверьте его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$.

3 Теоретическое введение

Организация стека

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).

Добавление элемента в стек

Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек. Существует ещё две команды для добавления значений в стек. Это команда pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ax, cx, dx, bx, sp, bp, si, di.

А также команда `pushf`, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов.

Извлечение элемента из стека

Команда `pop` извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр `esp`, после этого уменьшает значение регистра `esp` на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек. Аналогично команде записи в стек существует команда `popa`, которая восстанавливает из стека все регистры общего назначения, и команда `popf` для перемещения значений из вершины стека в регистр флагов.

Инструкции организации циклов

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре `ecx`. Наиболее простой является инструкция `loop`. Она позволяет организовать безусловный цикл. Инструкция `loop` выполняется в два этапа. Сначала из регистра `ecx` вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды `loop`.

4 Выполнение лабораторной работы

1. Создали каталог для программ лабораторной работы № 8, перешли в него и создали файл lab8-1.asm (рис. 4.1 Создание каталога и файла для выполнения лабораторной работы).

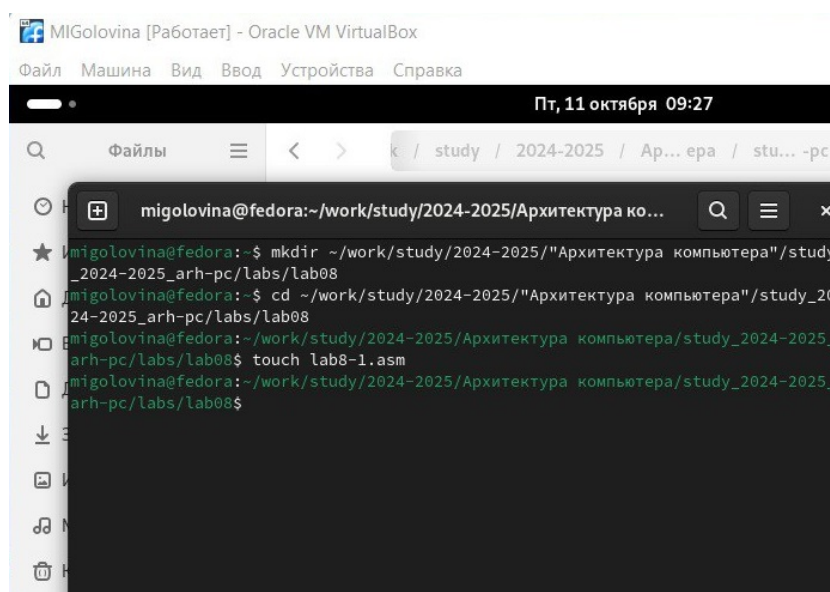
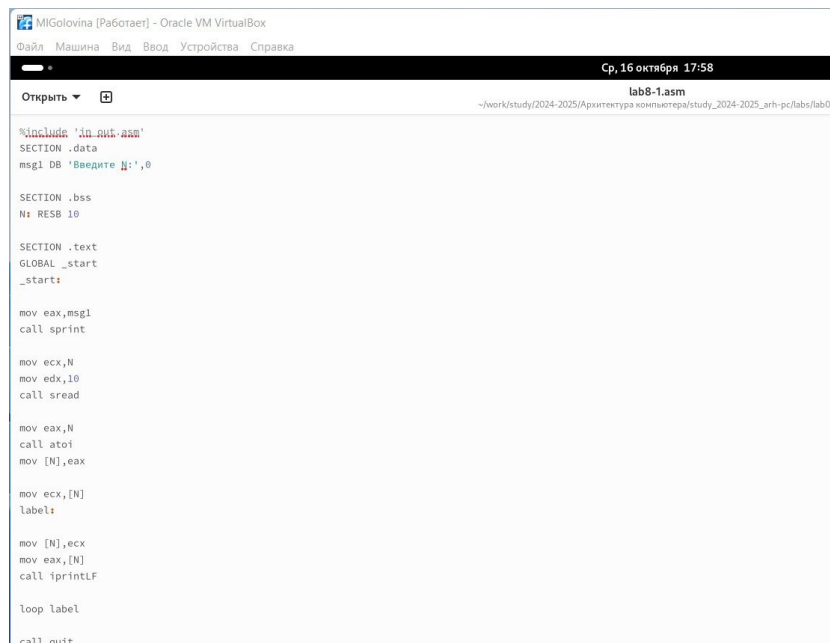


Рис. 4.1: Создание каталога и файла для выполнения лабораторной работы

2. Ввели в файл lab8-1.asm текст программы из листинга 1 (рис. 4.2 Листинг 1) методического указания. Создали исполняемый файл и запустили его. Посмотрели результаты работы (рис. 4.3 Результаты работы программы из листинга 1 методического указания).



```
%include "io_out.asm"
SECTION .data
msg1 DB "Введите N:",0

SECTION .bss
N: RESB 10

SECTION .text
GLOBAL _start
_start:

mov eax,msg1
call sprintf

mov ecx,N
mov edx,10
call sread

mov eax,N
call atoi
mov [N],eax

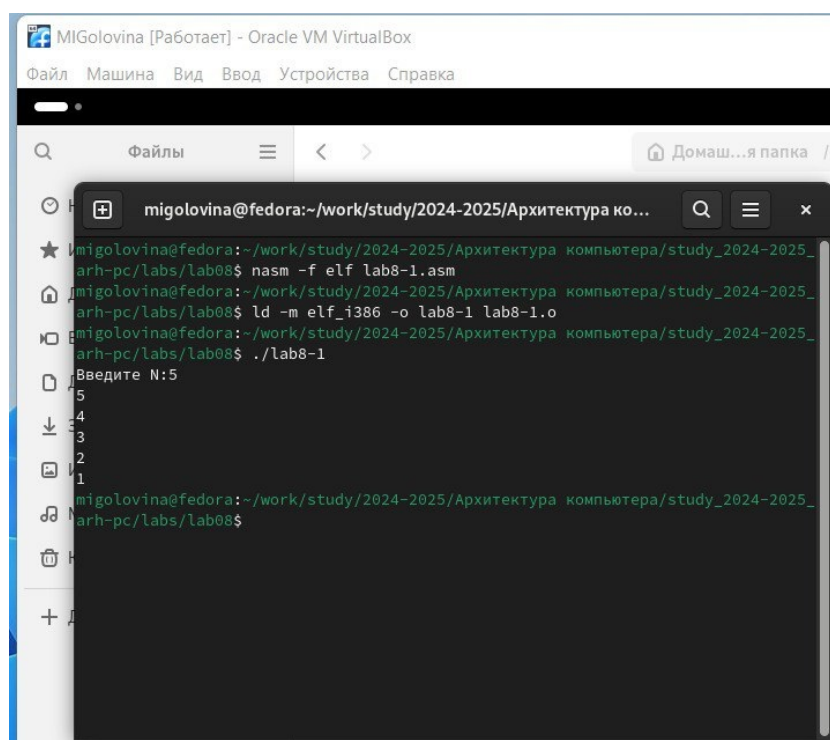
mov ecx,[N]
label:

mov [N],ecx
mov eax,[N]
call iprintLF

loop label

call quit
```

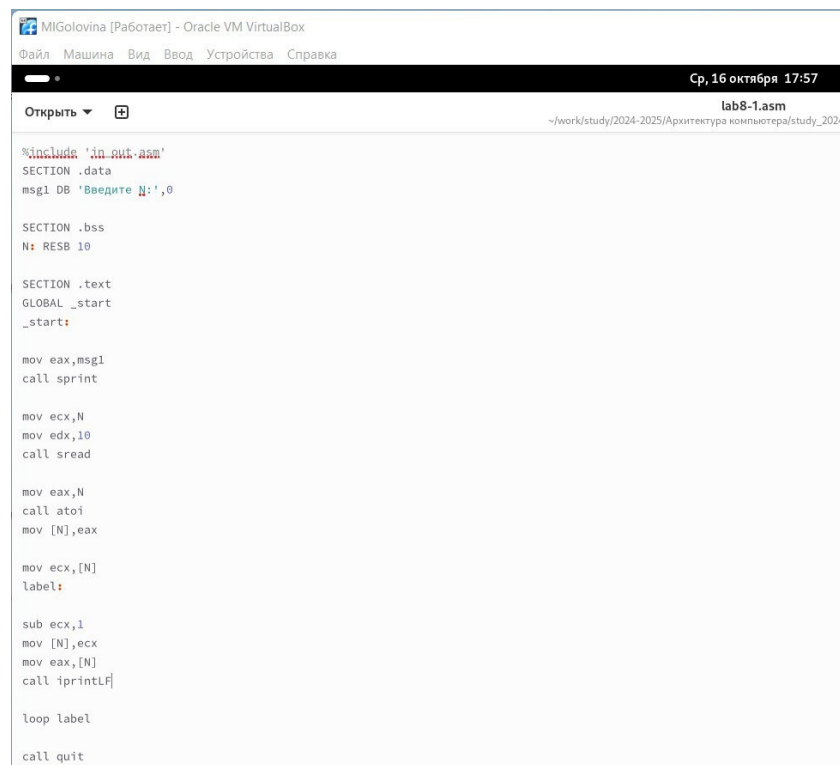
Рис. 4.2: Листинг 1



```
migolovina@fedora:~/work/study/2024-2025/Архитектура ко...
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$ nasm -f elf lab8-1.asm
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$ ./lab8-1
Введите N:5
5
4
3
2
1
120
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$
```

Рис. 4.3: Результаты работы программы из листинга 1 методического указания

3. Изменили текст программы добавив изменение значение регистра есх в цикле (рис.4.4 Листинг 1 с внесенными изменениями согласно методическому указанию). Создали исполняемый файл и запустили его. Посмотрели результаты работы (рис. 4.5 Результаты работы программы из Листинга 1 с внесенными изменениями согласно методическому указанию).



```
%include 'in-out.asm'
SECTION .data
msg1 DB 'Введите N:',0

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
```

Рис. 4.4: Листинг 1 с внесенными изменениями согласно методическому указанию

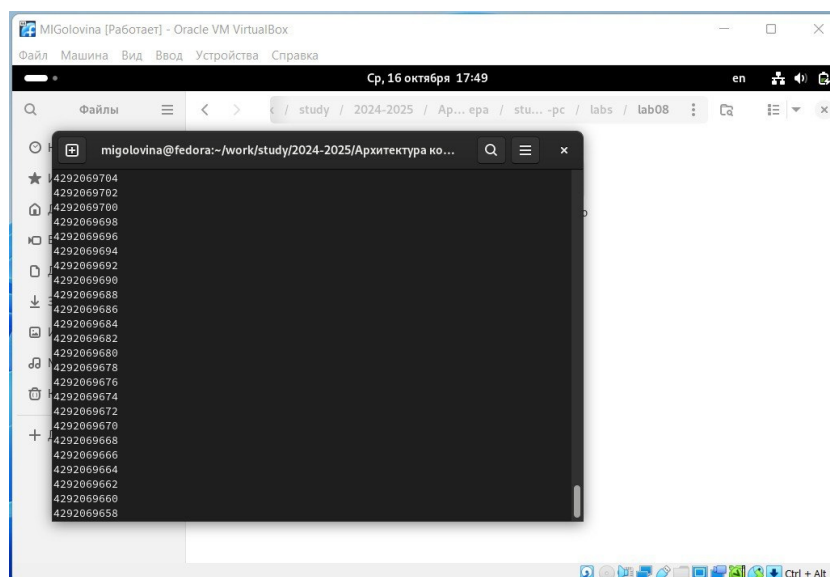


Рис. 4.5: Результаты работы программы из Листинга 1 с внесенными изменениями согласно методическому указанию

Цикл закольцевался и стал бесконечным.

4. Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Внесли изменения в текст программы добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop` (рис. 4.6 Измененный Листинг 1 с внесенными командами `push` и `pop`). Создали исполняемый файл и проверили его работу (рис. 4.7 Результаты работы программы измененного Листинга 1 с внесенными командами `push` и `pop`).

```
%include 'in_out.asm'
SECTION .data
msg1 DB 'Введите N:',0

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:

push ecx
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintf
pop ecx

loop label

call quit
```

Рис. 4.6: Измененный Листинг 1 с внесенными командами push и pop

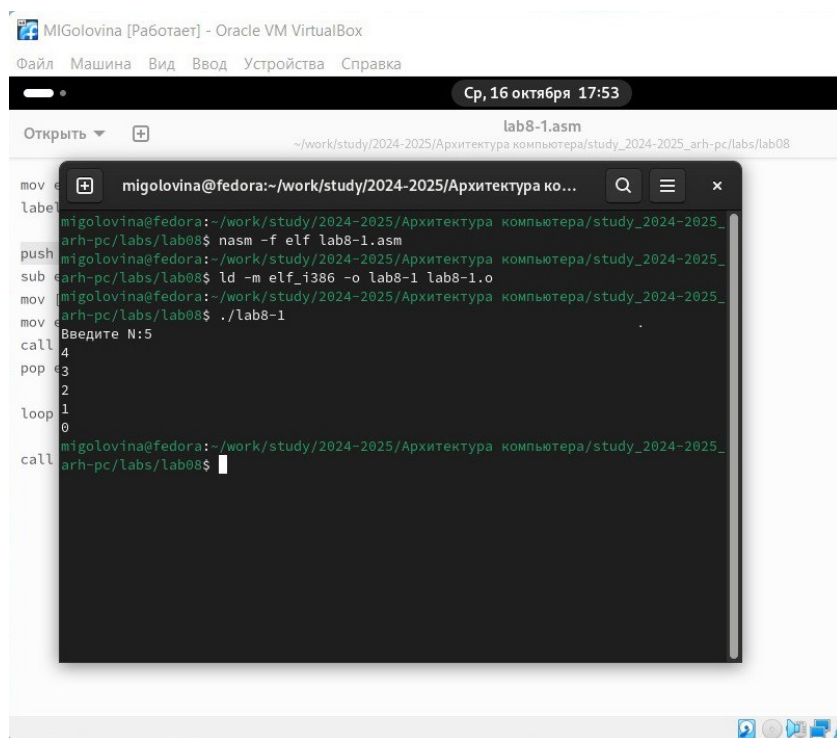


Рис. 4.7: Результаты работы программы измененного Листинга 1 с внесенными командами `push` и `pop`

Цикл и счетчик отработал правильно. По итогу после изменения программы, число проходки циклов стал соответствовать числу введенному с клавиатуры.

5. Создали файл `lab8-2.asm` и ввели в него текст программы из листинга 2 методического указания (рис. 4.8 Листинг 2). Создали исполняемый файл и запустили его, указав необходимые аргументы. Программа выводит все 3 аргумента, которые ввели, но в разной вариации (рис. 4.9 Результат работы программы из Листинга 2).



```
%include 'in-out.asm'
SECTION .text
GLOBAL _start

_start:
pop ecx

pop ecx

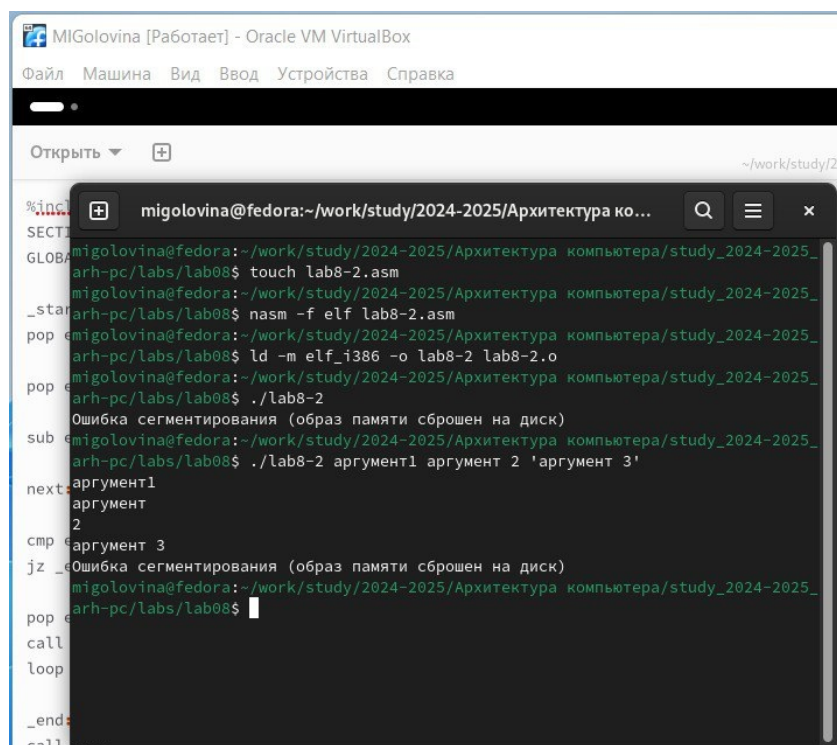
sub ecx,1

next:
cmp ecx,0
jz _end

pop eax
call sprintf
loop next

_end:
call quit
```

Рис. 4.8: Листинг 2

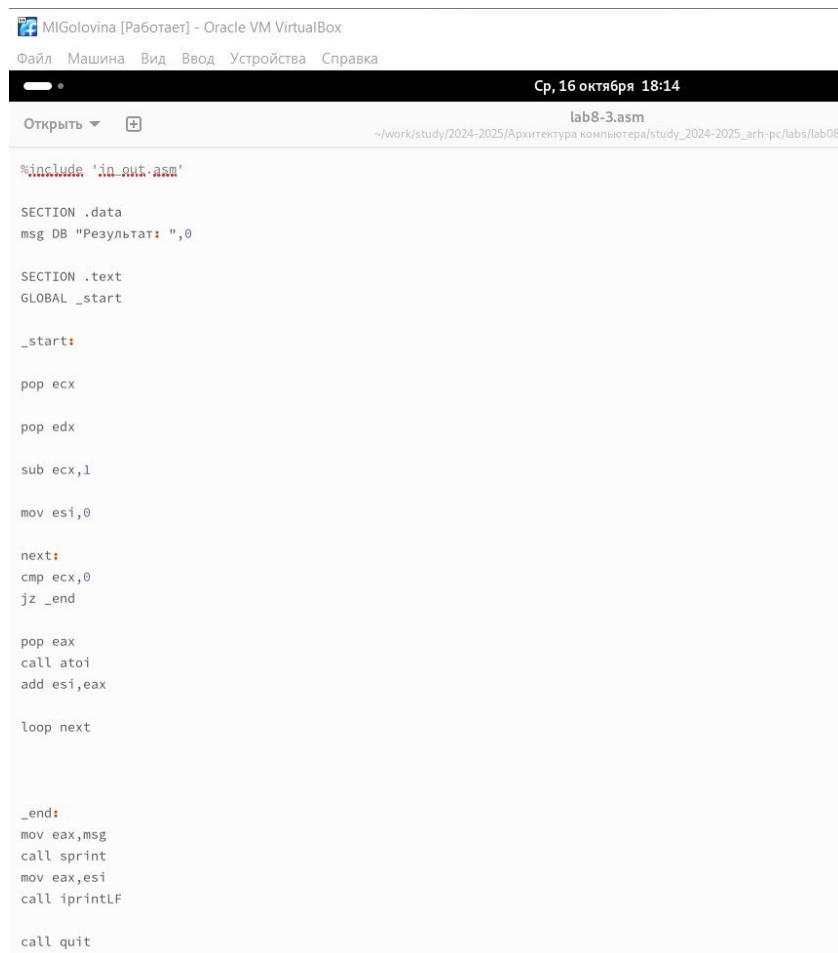


```
migolovina@fedora:~/work/study/2024-2025/Архитектура ко...
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$ touch lab8-2.asm
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$ nasm -f elf lab8-2.asm
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$ ./lab8-2
next
quit
```

Рис. 4.9: Результат работы программы из Листинга 2

6. Создали файл lab8-3.asm и ввели в него текст программы из листинга 3 методического указания (рис. 4.10 Листинг 3). Создали исполняемый файл и запустили его, указав необходимые аргументы. Программа вывела сумму

чисел, которые мы ввели (рис. 4.11 Результат работы программы из Листинга 3).



```
%include 'in_out.asm'

SECTION .data
msg DB "Результат: ",0

SECTION .text
GLOBAL _start

_start:

pop ecx

pop edx

sub ecx,1

mov esi,0

next:
cmp ecx,0
jz _end

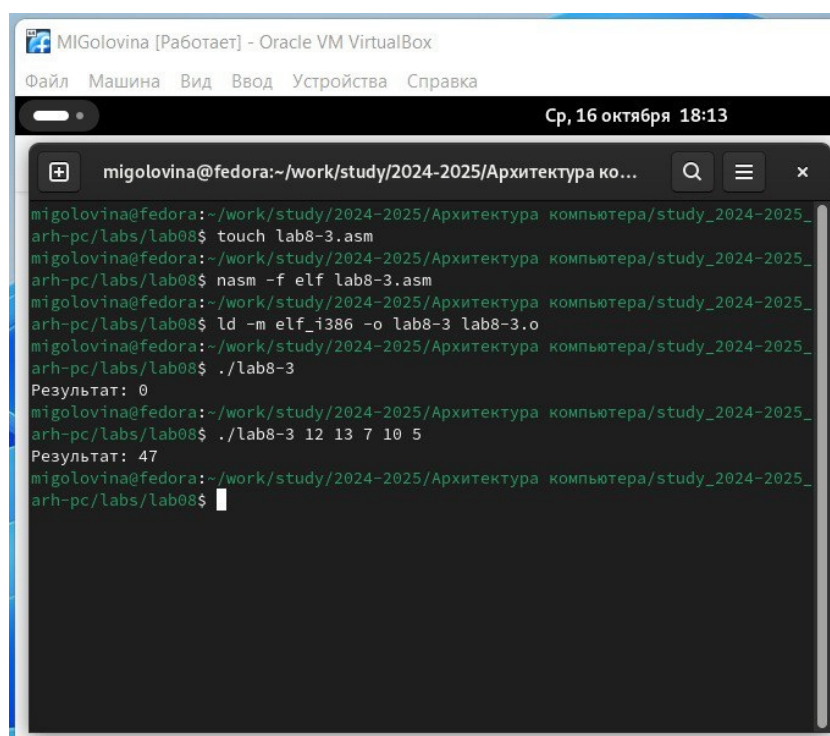
pop eax
call atoi
add esi,eax

loop next

_end:
mov eax,msg
call sprint
mov eax,esi
call iprintLF

call quit
```

Рис. 4.10: Листинг 3



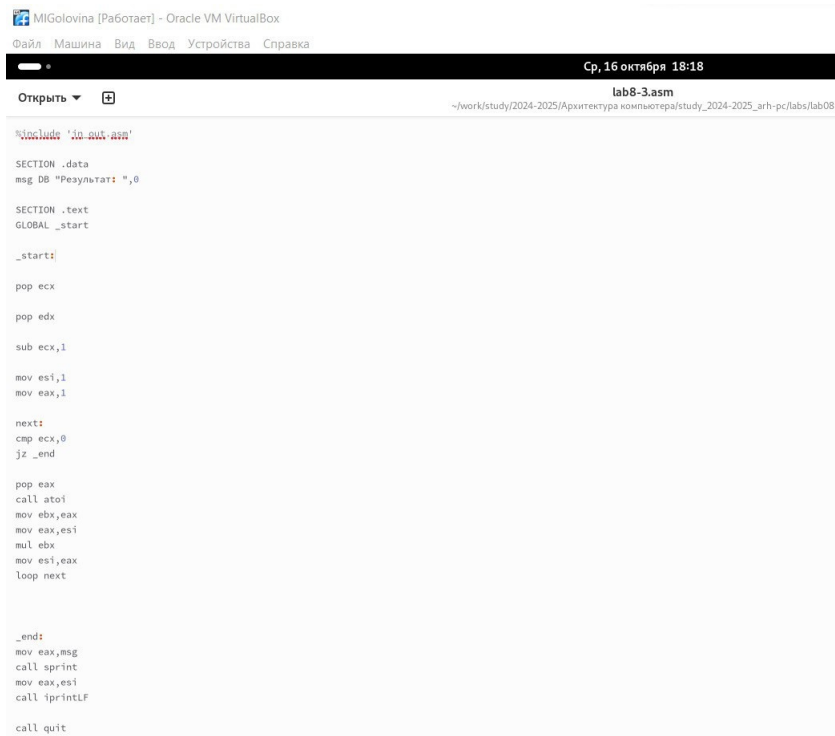
The screenshot shows a terminal window titled "migolovina@fedora:~/work/study/2024-2025/Архитектура ко...". The terminal displays the following commands and their outputs:

```

migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ touch lab8-3.asm
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ nasm -f elf lab8-3.asm
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ ./lab8-3
Результат: 0
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$
```

Рис. 4.11: Результат работы программы из Листинга 3

Изменили текст Листинга 3 для вычисления произведения аргументов командной строки (рис. 4.12 Измененный Листинг 3 для вычисления произведения аргументов командной строки). Проверили результаты работы программы (рис. 4.13 Результаты работы программы для вычисления произведения аргументов командной строки).



```
%include 'in_out.asm'

SECTION .data
msg DB "Результат: ",0

SECTION .text
GLOBAL _start

_start:

pop ecx

pop edx

sub ecx,1

mov esi,1
mov eax,1

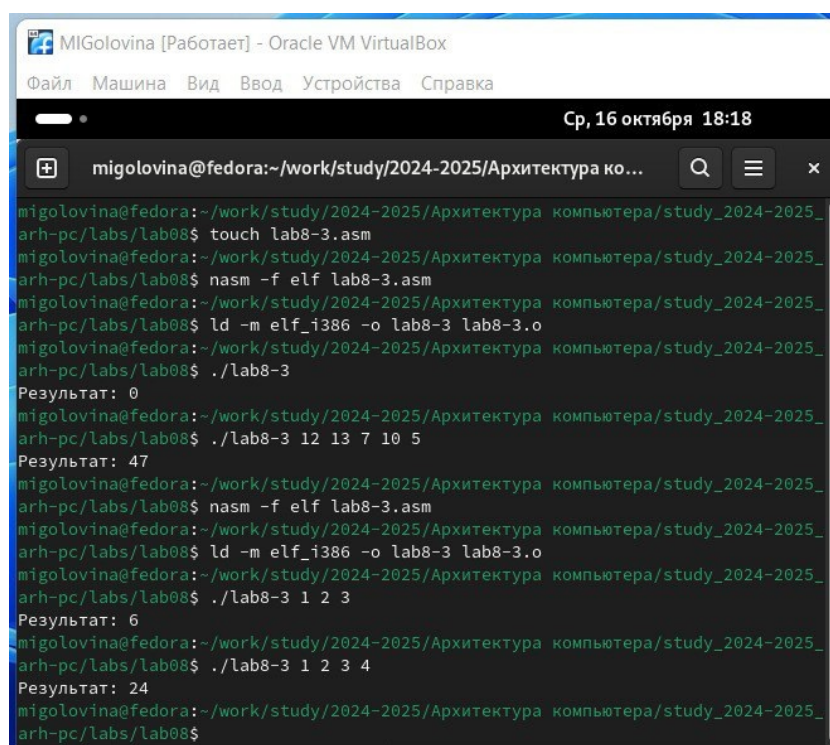
next:
cmp ecx,0
jz _end

pop eax
call atoi
mov ebx,eax
mov eax,esi
mul ebx
mov esi,eax
loop next

_end:
mov eax,msg
call sprintf
mov eax,esi
call !printfLF

call quit
```

Рис. 4.12: Измененный Листинг 3 для вычисления произведения аргументов командной строки



The screenshot shows a terminal window titled "migolovina@fedora:~/work/study/2024-2025/Архитектура ко...". The terminal displays the following commands and outputs:

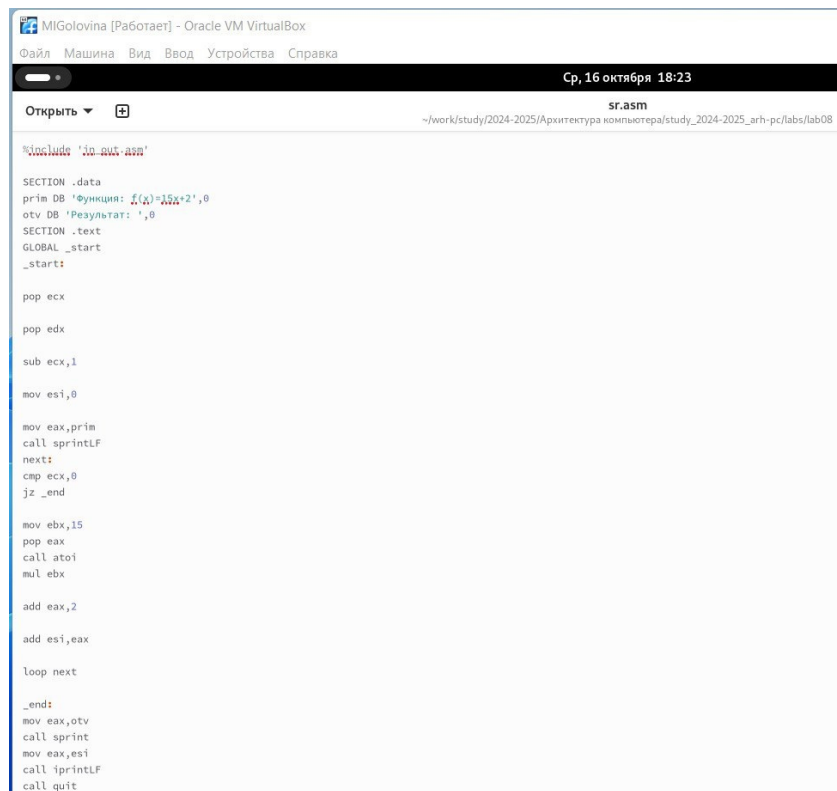
```
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ touch lab8-3.asm
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ nasm -f elf lab8-3.asm
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ ./lab8-3
Результат: 0
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ nasm -f elf lab8-3.asm
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ ./lab8-3 1 2 3
Результат: 6
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$ ./lab8-3 1 2 3 4
Результат: 24
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-pc/labs/lab08$
```

Рис. 4.13: Результаты работы программы для вычисления произведения аргументов командной строки

Задание для самостоятельной работы

1. Напишите программу, которая находит сумму значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$, т.е. программа должна выводить значение $f(x_1) + f(x_2) + \dots + f(x_n)$. Значения x_i передаются как аргументы. Вид функции $f(x)$ выбрать из таблицы вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Создайте исполняемый файл и проверьте его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$.

При выполнении лабораторной работы №6 у меня получился вариант 11, соответственно для варианта №11 $f(x) = 15x + 2$.



```
%include 'io_out.asm'

SECTION .data
prim DB 'Функция: f(x)=15x+2',0
otv DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

pop ecx

pop edx

sub ecx,1

mov esi,0

mov eax,prim
call sprintf
next:
cmp ecx,0
jz _end

mov ebx,15
pop eax
call atoi
mul ebx

add eax,2

add esi,eax

loop next

_end:
mov eax,otv
call sprintf
mov eax,esi
call iprintf
call quit
```

Рис. 4.14: Листинг самостоятельного задания №1

```
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$ nasm -f elf sr.asm
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$ ld -m elf_i386 -o sr sr.o
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$ ./sr 1 2 3
Функция: f(x)=15x+2
Результат: 96
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$ ./sr 1 2 3 4
Функция: f(x)=15x+2
Результат: 158
migolovina@fedora:~/work/study/2024-2025/Архитектура компьютера/study_2024-2025_
arh-pc/labs/lab08$
```

Рис. 4.15: Результаты работы программы по самостоятельному заданию №1

5 Выводы

Приобрели 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).