

Отчет по лабораторной работе №8

Дисциплина: архитектура компьютера

Панина Жанна Валерьевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация циклов в NASM	8
4.2	Обработка аргументов командной строки	12
4.3	Задание для самостоятельной работы	17
5	Выводы	20
6	Список литературы	21

Список иллюстраций

4.1	Создание каталога	8
4.2	Копирование программы из листинга	9
4.3	Запуск программы	9
4.4	Изменение программы	10
4.5	Запуск измененной программы	11
4.6	Добавление push и pop в цикл программы	11
4.7	Запуск измененной программы	12
4.8	Копирование программы из листинга	13
4.9	Запуск второй программы	14
4.10	Копирование программы из третьего листинга	14
4.11	Запуск третьей программы	15
4.12	Изменение третьей программы	16
4.13	Запуск измененной третьей программы	16
4.14	Написание программы для самостоятельной работы	17
4.15	Запуск программы для самостоятельной работы	19

Список таблиц

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

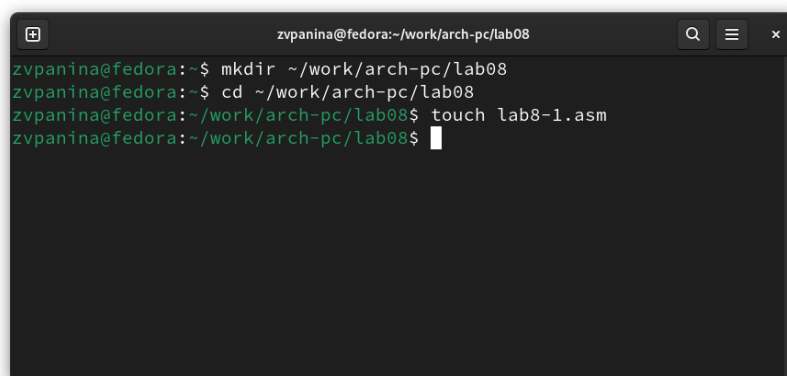
3 Теоретическое введение

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.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

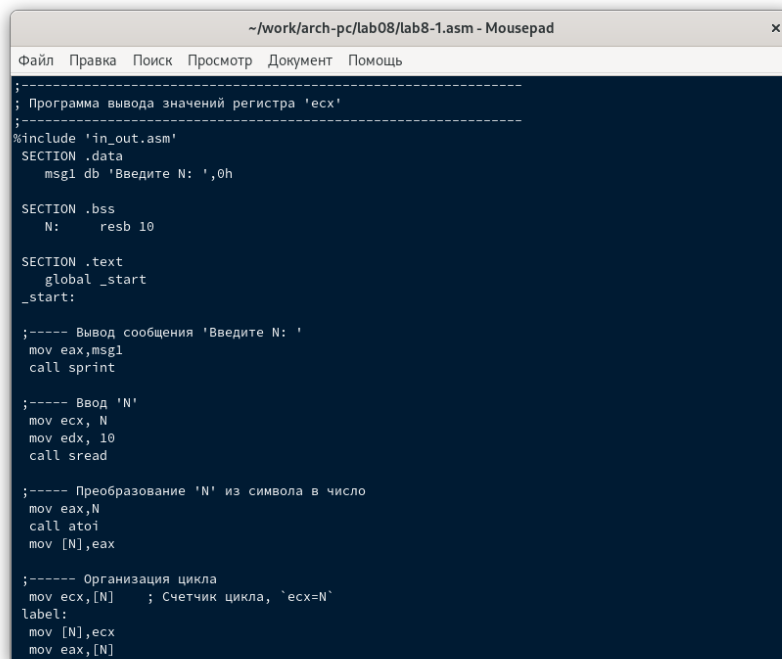
Создаю каталог для программ лабораторной работы №8, перехожу в него и создаю файл lab8-1.asm(рис. -fig. 4.1).



```
zvpanina@fedora:~/work/arch-pc/lab08
zvpanina@fedora:~$ mkdir ~/work/arch-pc/lab08
zvpanina@fedora:~$ cd ~/work/arch-pc/lab08
zvpanina@fedora:~/work/arch-pc/lab08$ touch lab8-1.asm
zvpanina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.1: Создание каталога

Копирую в созданный файл программу из листинга. (рис. -fig. 4.2).



```
~/work/arch-pc/lab08/lab8-1.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь
-----
; Программа вывода значений регистра 'ecx'
-----
%include 'in_out.asm'
SECTION .data
    msg1 db 'Введите N: ',0h

SECTION .bss
    N:    resb 10

SECTION .text
    global _start
    _start:

;---- Вывод сообщения 'Введите N: '
    mov eax,msg1
    call sprint

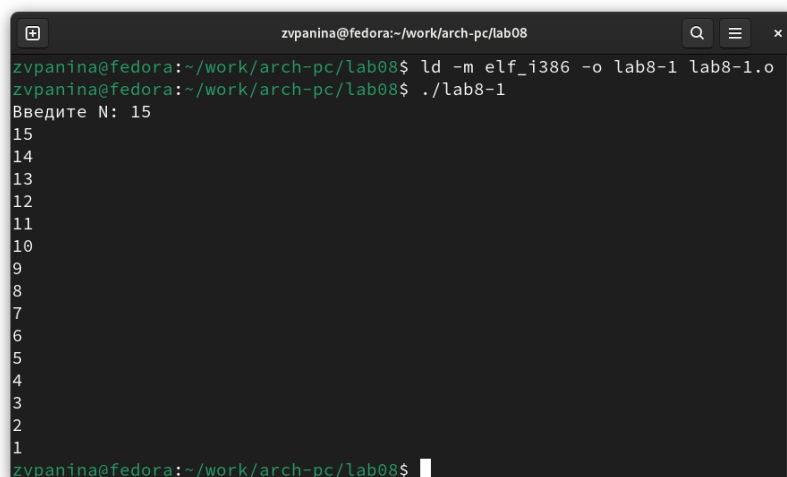
;---- Ввод 'N'
    mov ecx, N
    mov edx, 10
    call sread

;---- Преобразование 'N' из символа в число
    mov eax,N
    call atoi
    mov [N],eax

;----- Организация цикла
    mov ecx,[N]    ; Счетчик цикла, 'ecx=N'
label:
    mov [N],ecx
    mov eax,[N]
```

Рис. 4.2: Копирование программы из листинга

Создаю исполняемый файл и проверяю его работу (рис. -fig. 4.3). Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы.

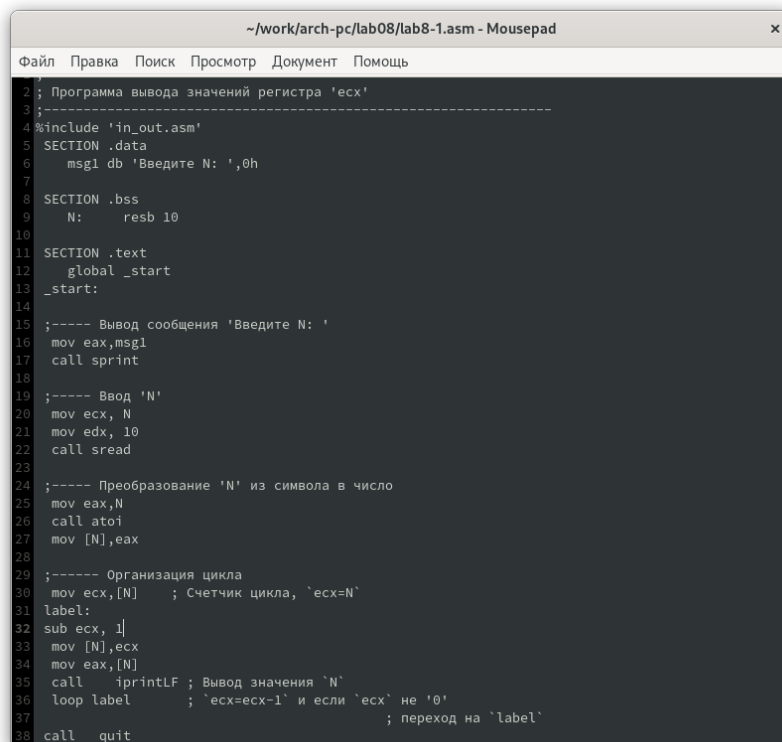


```
zvpanina@fedora:~/work/arch-pc/lab08
zvpanina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
zvpanina@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 15
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
zvpanina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.3: Запуск программы

Изменяю текст программы, добавив изменение значения регистра `ecx` в цикле

(рис. -fig. 4.4).

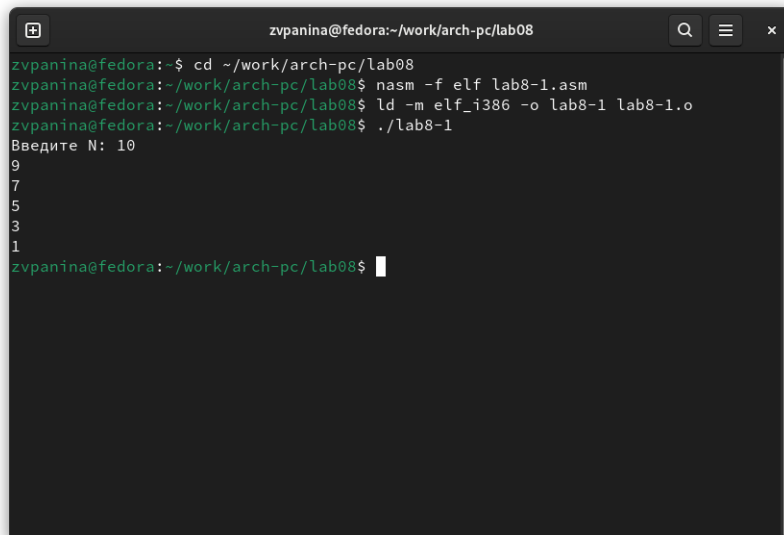


```
~/work/arch-pc/lab08/lab8-1.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь

2 ; Программа вывода значений регистра 'ecx'
3 ;
4 %include 'in_out.asm'
5 SECTION .data
6     msg1 db 'Введите N: ',0h
7
8 SECTION .bss
9     N:    resb 10
10
11 SECTION .text
12     global _start
13     _start:
14
15 ;---- Вывод сообщения 'Введите N: '
16     mov eax,msg1
17     call sprint
18
19 ;---- Ввод 'N'
20     mov ecx,N
21     mov edx,10
22     call sread
23
24 ;---- Преобразование 'N' из символа в число
25     mov eax,N
26     call atoi
27     mov [N],eax
28
29 ;---- Организация цикла
30     mov ecx,[N]    ; Счетчик цикла, 'ecx=N'
31 label:
32     sub ecx, 1
33     mov [N],ecx
34     mov eax,[N]
35     call iprintfLF ; Вывод значения 'N'
36     loop label    ; 'ecx=ecx-1' и если 'ecx' не '0'
37                   ; переход на 'label'
38     call quit
```

Рис. 4.4: Изменение программы

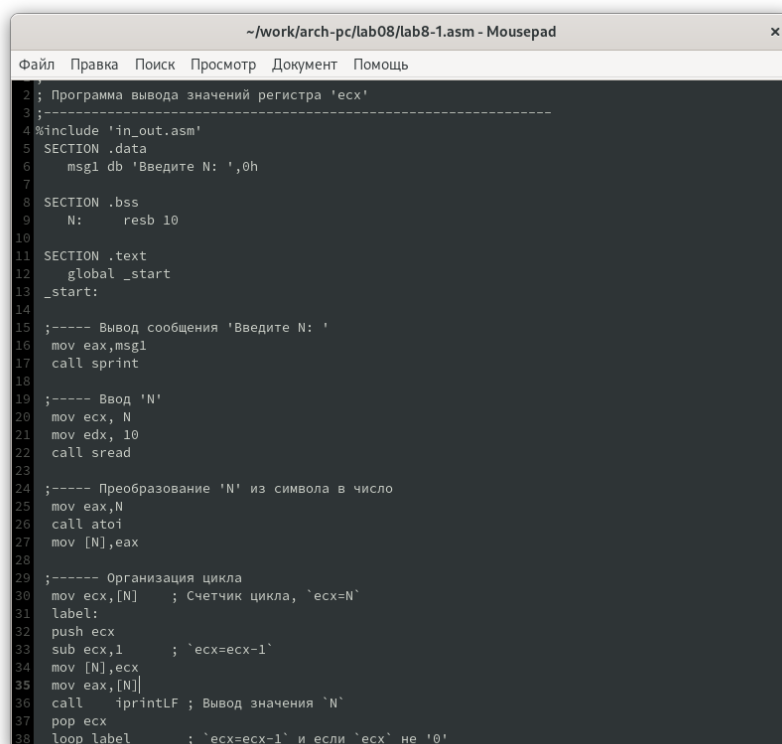
Из-за того, что теперь регистр ecx на каждой итерации уменьшается на 2 значения, количество итераций уменьшается вдвое (рис. -fig. 4.5).



```
zvpanina@fedora:~/work/arch-pc/lab08
zvpanina@fedora:~/work/arch-pc/lab08$ cd ~/work/arch-pc/lab08
zvpanina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
zvpanina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
zvpanina@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
7
5
3
1
zvpanina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.5: Запуск измененной программы

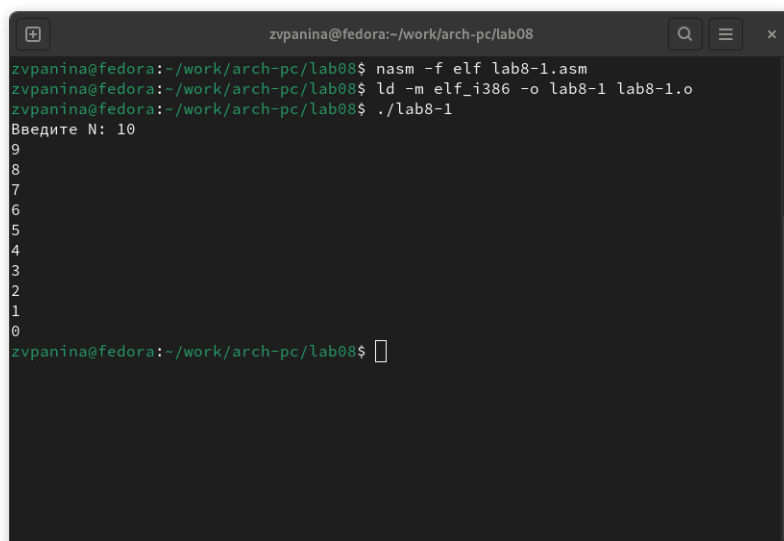
Добавляю команды push и pop в программу (рис. -fig. 4.6).



```
~/work/arch-pc/lab08/lab8-1.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь
1 ;
2 ; Программа вывода значений регистра 'ecx'
3 ;
4 %include 'in_out.asm'
5 SECTION .data
6     msg1 db 'Введите N: ',0h
7
8 SECTION .bss
9     N:    resb 10
10
11 SECTION .text
12     global _start
13     _start:
14
15 ;---- Вывод сообщения 'Введите N: '
16     mov eax,msg1
17     call sprint
18
19 ;---- Ввод 'N'
20     mov ecx, N
21     mov edx, 10
22     call sread
23
24 ;---- Преобразование 'N' из символа в число
25     mov eax,N
26     call atoi
27     mov [N],eax
28
29 ;----- Организация цикла
30     mov ecx,[N]    ; Счетчик цикла, 'ecx=N'
31     label:
32     push ecx
33     sub ecx,1      ; 'ecx=ecx-1'
34     mov [N],ecx
35     mov eax,[N]
36     call iprintLF ; Вывод значения 'N'
37     pop ecx
38     loop label     ; 'ecx=ecx-1' и если 'ecx' не '0'
```

Рис. 4.6: Добавление push и pop в цикл программы

Теперь количество итераций соответствует значению N, введенному с клавиатуры, но произошло смещение выводимых чисел на -1 (рис. -fig. 4.7).

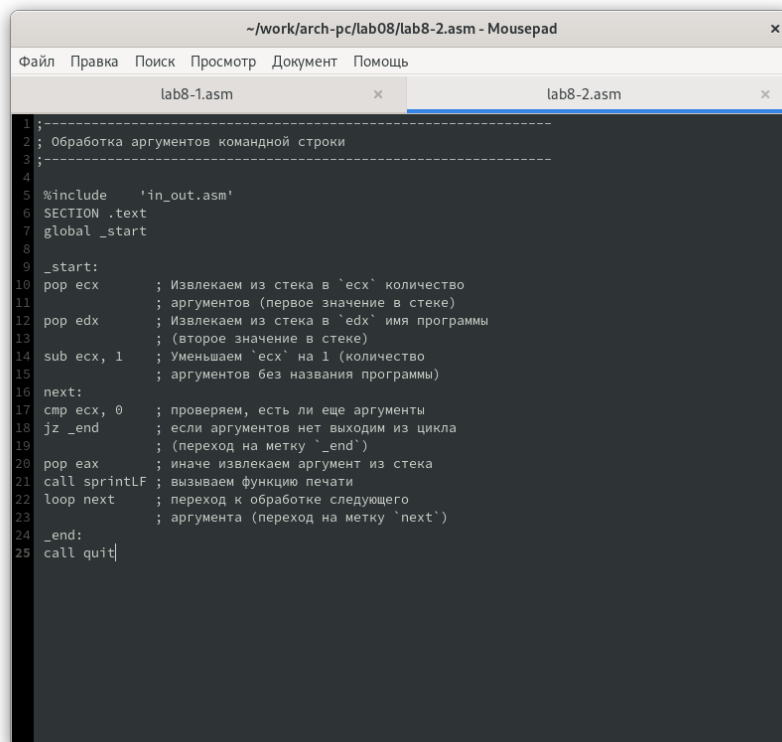


```
zvpanina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
zvpanina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
zvpanina@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
zvpanina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.7: Запуск измененной программы

4.2 Обработка аргументов командной строки

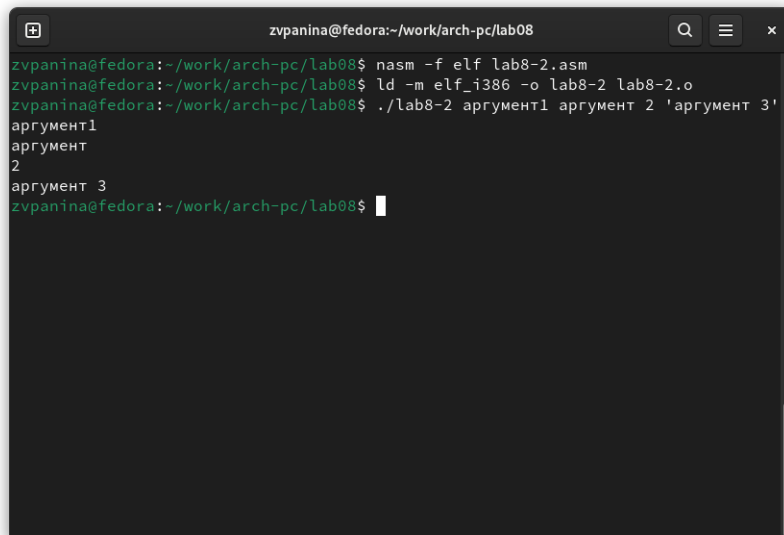
Создаю файл lab8-2.asm и копирую в него код из следующего листинга (рис. -fig. 4.8).



```
1 ;-----
2 ; Обработка аргументов командной строки
3 ;-----
4
5 %include 'in_out.asm'
6 SECTION .text
7 global _start
8
9 _start:
10 pop ecx      ; Извлекаем из стека в 'ecx' количество
11              ; аргументов (первое значение в стеке)
12 pop edx      ; Извлекаем из стека в 'edx' имя программы
13              ; (второе значение в стеке)
14 sub ecx, 1    ; Уменьшаем 'ecx' на 1 (количество
15              ; аргументов без названия программы)
16 next:
17 cmp ecx, 0    ; проверяем, есть ли еще аргументы
18 jz _end       ; если аргументов нет выходим из цикла
19              ; (переход на метку '_end')
20 pop eax       ; иначе извлекаем аргумент из стека
21 call sprintf  ; вызываем функцию печати
22 loop next     ; переход к обработке следующего
23              ; аргумента (переход на метку 'next')
24 _end:
25 call quit
```

Рис. 4.8: Копирование программы из листинга

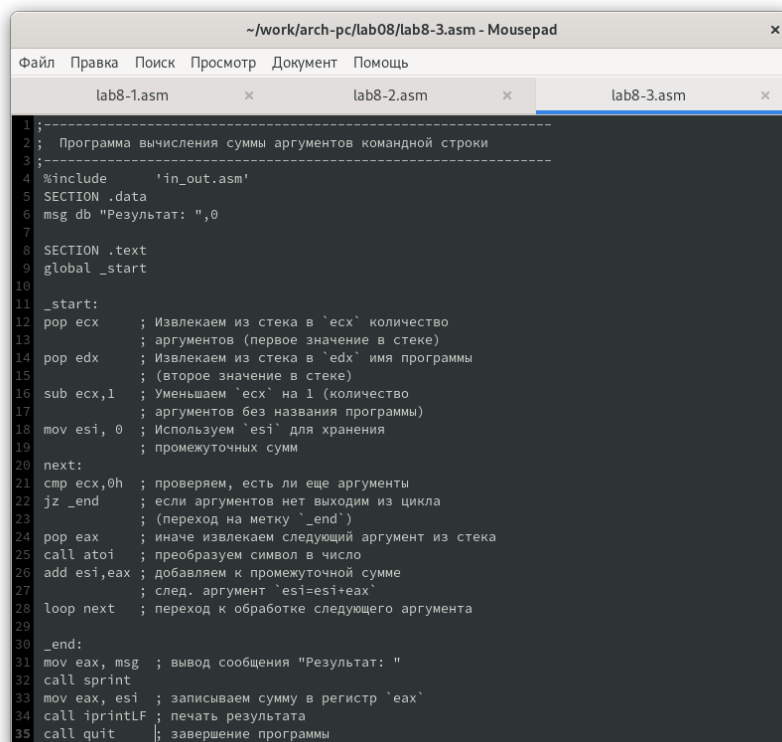
Компилирую программу и запускаю, указав аргументы. Программой было обработано то же количество аргументов, что и было введено, но слова “аргумент” и “2” были приняты как два аргумента, поскольку были записаны через пробел, т.е. у меня 4 аргумента (рис. -fig. 4.9).



```
zvpanina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
zvpanina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
zvpanina@fedora:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
zvpanina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.9: Запуск второй программы

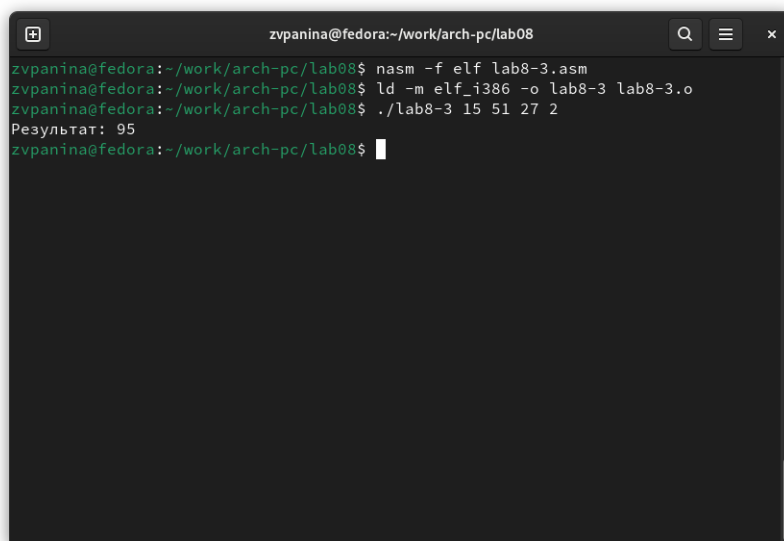
Создаю новый файл для программы и копирую в него код из третьего листинга (рис. -fig. 4.10).



```
1 ;-----
2 ; Программа вычисления суммы аргументов командной строки
3 ;-----
4 %include 'in_out.asm'
5 SECTION .data
6 msg db "Результат: ",0
7
8 SECTION .text
9 global _start
10
11 _start:
12 pop ecx ; Извлекаем из стека в 'ecx' количество
13 ; аргументов (первое значение в стеке)
14 pop edx ; Извлекаем из стека в 'edx' имя программы
15 ; (второе значение в стеке)
16 sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
17 ; аргументов без названия программы)
18 mov esi, 0 ; Используем 'esi' для хранения
19 ; промежуточных сумм
20 next:
21 cmp ecx,0h ; проверяем, есть ли еще аргументы
22 jz _end ; если аргументов нет выходим из цикла
23 ; (переход на метку '_end')
24 pop eax ; иначе извлекаем следующий аргумент из стека
25 call atoi ; преобразуем символ в число
26 add esi,eax ; добавляем к промежуточной сумме
27 ; след. аргумент 'esi=esi+eax'
28 loop next ; переход к обработке следующего аргумента
29
30 _end:
31 mov eax, msg ; вывод сообщения "Результат: "
32 call sprint
33 mov eax, esi ; записываем сумму в регистр 'eax'
34 call iprintLF ; печать результата
35 call quit ; завершение программы
```

Рис. 4.10: Копирование программы из третьего листинга

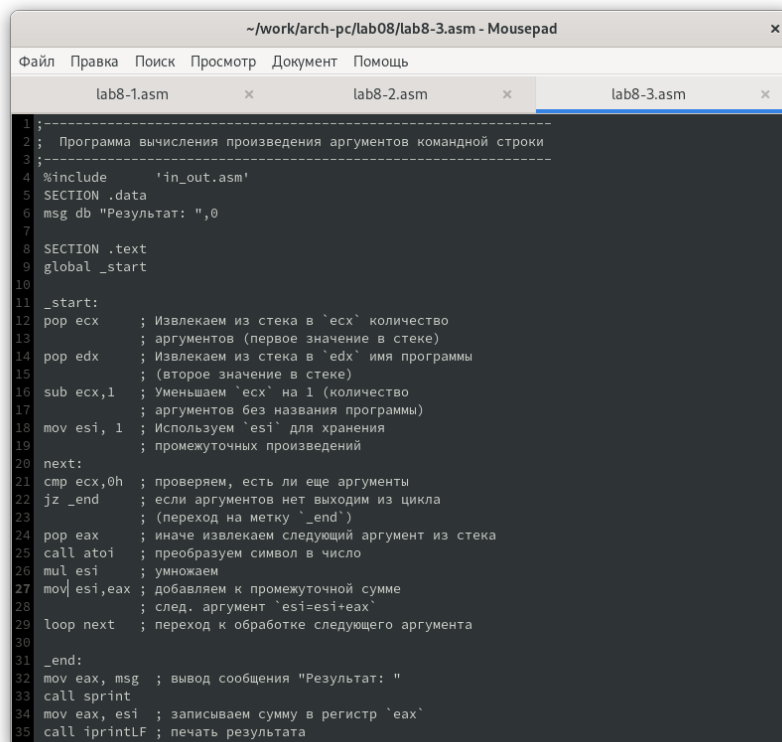
Компилирую программу и запускаю, указав в качестве аргументов некоторые числа, программа их складывает (рис. -fig. 4.11).

A terminal window titled 'zvpanina@fedora:~/work/arch-pc/lab08' with search, menu, and close icons in the title bar. The terminal shows the following commands and output:

```
zvpanina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
zvpanina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
zvpanina@fedora:~/work/arch-pc/lab08$ ./lab8-3 15 51 27 2
Результат: 95
zvpanina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.11: Запуск третьей программы

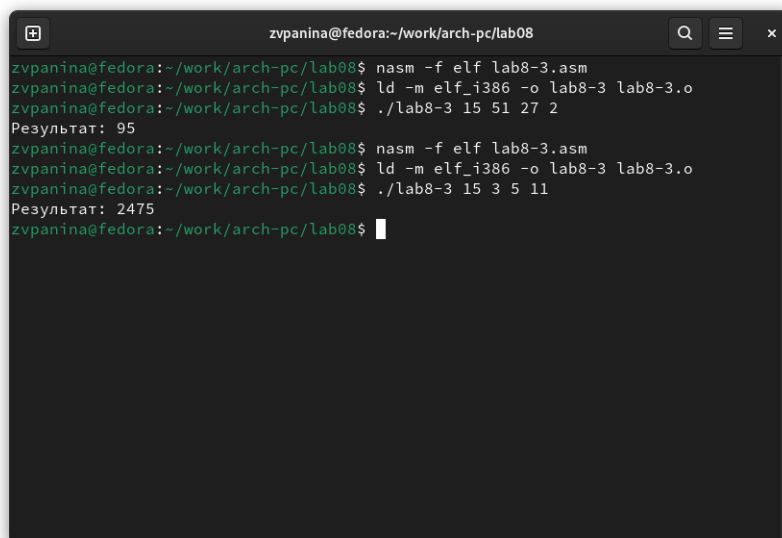
Изменяю текст программы для вычисления произведения аргументов (рис. -fig. 4.12).



```
1 ;-----
2 ; Программа вычисления произведения аргументов командной строки
3 ;-----
4 %include 'in_out.asm'
5 SECTION .data
6 msg db "Результат: ",0
7
8 SECTION .text
9 global _start
10
11 _start:
12 pop ecx ; Извлекаем из стека в 'ecx' количество
13 ; аргументов (первое значение в стеке)
14 pop edx ; Извлекаем из стека в 'edx' имя программы
15 ; (второе значение в стеке)
16 sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
17 ; аргументов без названия программы)
18 mov esi, 1 ; Используем 'esi' для хранения
19 ; промежуточных произведений
20 next:
21 cmp ecx,0h ; проверяем, есть ли еще аргументы
22 jz _end ; если аргументов нет выходим из цикла
23 ; (переход на метку '_end')
24 pop eax ; иначе извлекаем следующий аргумент из стека
25 call atoi ; преобразуем символ в число
26 mul esi ; умножаем
27 mov esi,eax ; добавляем к промежуточной сумме
28 ; след. аргумент 'esi=esi+eax'
29 loop next ; переход к обработке следующего аргумента
30
31 _end:
32 mov eax, msg ; вывод сообщения "Результат: "
33 call sprint
34 mov eax, esi ; записываем сумму в регистр 'eax'
35 call iprintLF ; печать результата
```

Рис. 4.12: Изменение третьей программы

Программа действительно теперь умножает данные на вход числа (рис. -fig. 4.13).

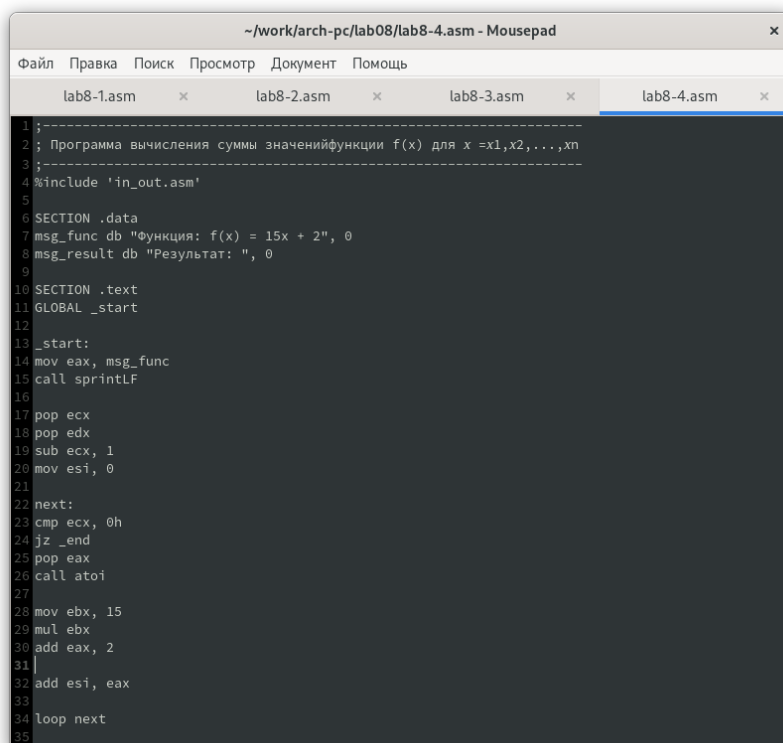


```
zvpanina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
zvpanina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
zvpanina@fedora:~/work/arch-pc/lab08$ ./lab8-3 15 51 27 2
Результат: 95
zvpanina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
zvpanina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
zvpanina@fedora:~/work/arch-pc/lab08$ ./lab8-3 15 3 5 11
Результат: 2475
zvpanina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.13: Запуск измененной третьей программы

4.3 Задание для самостоятельной работы

У меня 11 вариант, поэтому пишу программу, которая будет находить сумму значений для функции $f(x) = 15x + 2$ (рис. -fig. 4.14).



```
~/work/arch-pc/lab08/lab8-4.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь
lab8-1.asm  x  lab8-2.asm  x  lab8-3.asm  x  lab8-4.asm  x
1 ;
2 ; Программа вычисления суммы значений функции f(x) для x = x1, x2, ..., xn
3 ;
4 %include 'in_out.asm'
5
6 SECTION .data
7 msg_func db "Функция: f(x) = 15x + 2", 0
8 msg_result db "Результат: ", 0
9
10 SECTION .text
11 GLOBAL _start
12
13 _start:
14 mov eax, msg_func
15 call sprintf
16
17 pop ecx
18 pop edx
19 sub ecx, 1
20 mov esi, 0
21
22 next:
23 cmp ecx, 0h
24 jz _end
25 pop eax
26 call atoi
27
28 mov ebx, 15
29 mul ebx
30 add eax, 2
31
32 add esi, eax
33
34 loop next
35
```

Рис. 4.14: Написание программы для самостоятельной работы

Код программы:

```
;-
; Программа вычисления суммы значений функции f(x) для x = x1, x2, ..., xn
;-
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 15x + 2", 0
msg_result db "Результат: ", 0
```

```

SECTION .text
GLOBAL _start

_start:
mov eax, msg_func
call sprintf

pop ecx
pop edx
sub ecx, 1
mov esi, 0

next:
cmp ecx, 0h
jz _end
pop eax
call atoi

mov ebx, 15
mul ebx
add eax, 2

add esi, eax

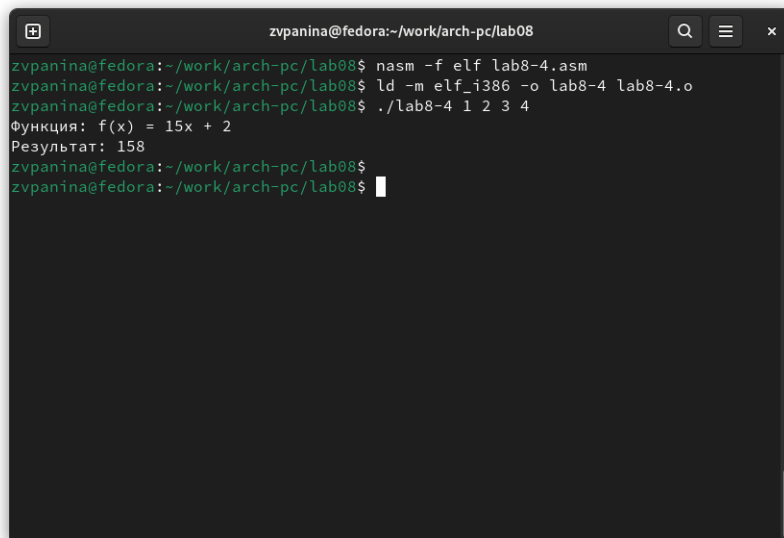
loop next

_end:
mov eax, msg_result

```

```
call sprint
mov eax, esi
call iprintLF
call quit
```

Проверяю работу программы, указав в качестве аргумента несколько чисел (рис. -fig. 4.15). Программа работает корректно.



```
zvpanina@fedora:~/work/arch-pc/lab08
zvpanina@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-4.asm
zvpanina@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-4 lab8-4.o
zvpanina@fedora:~/work/arch-pc/lab08$ ./lab8-4 1 2 3 4
Функция:  $f(x) = 15x + 2$ 
Результат: 158
zvpanina@fedora:~/work/arch-pc/lab08$
zvpanina@fedora:~/work/arch-pc/lab08$
```

Рис. 4.15: Запуск программы для самостоятельной работы

5 Выводы

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием циклов, а также научилась обрабатывать аргументы командной строки.

6 Список литературы

1. Курс на ТУИС
2. Лабораторная работа №8
3. Программирование на языке ассемблера NASM Столяров А. В.