

Отчёт по лабораторной работе 8

Дисциплина: архитектура компьютера

Давлетова Мадина

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	14
4.3	Выполнение заданий для самостоятельной работы	20
5	Выводы	23

Список иллюстраций

4.1	Изменение кода lab8-1.asm	9
4.2	Компиляция текста программы lab8-1.asm	10
4.3	Изменение кода lab8-1.asm	11
4.4	Компиляция текста программы lab8-1.asm	12
4.5	Изменение кода lab8-1.asm	13
4.6	Компиляция текста программы lab8-1.asm	14
4.7	Изменение кода lab8-2.asm	15
4.8	Компиляция текста программы lab8-2.asm	16
4.9	Изменение кода lab8-3.asm	17
4.10	Компиляция текста программы lab8-3.asm	18
4.11	Изменение кода lab8-3.asm	19
4.12	Компиляция текста программы lab8-3.asm	20
4.13	Изменение кода prog.asm	21
4.14	Компиляция текста программы prog.asm	22

Список таблиц

1 Цель работы

Целью работы является приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки..

2 Задание

1. Изучение циклов в ассемблере
2. Изучение стека в ассемблере
3. Изучение передачи аргументов в ассемблере
4. Выполнение заданий, рассмотрение примеров
5. Выполнение заданий для самостоятельной 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.

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.

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.

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.

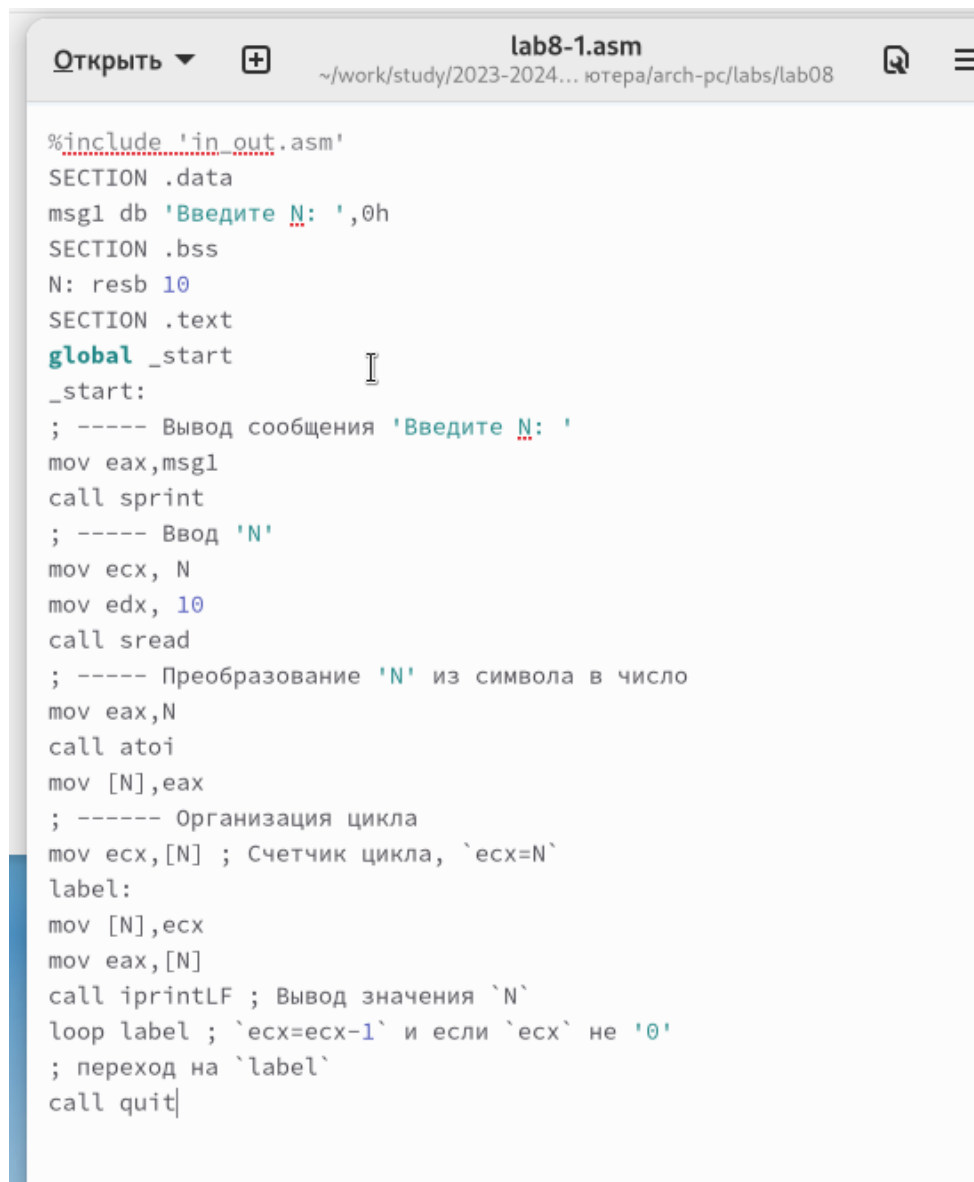
4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаю каталог для выполнения лабораторной работы № 8 и файл с именем lab8-1.asm.

При использовании инструкции `loop` в NASM для реализации циклов, нужно помнить о том, что данная инструкция использует регистр `ecx` в качестве счетчика и на каждой итерации уменьшает его значение на единицу.

Рассмотрим пример программы, которая выводит значение регистра `ecx`. Написала текст программы из листинга 8.1 в файле lab8-1.asm. (рис. [4.1])



```
lab8-1.asm
~/work/study/2023-2024... ютера/arch-pc/labs/lab08

%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]
call iprintLF ; Вывод значения `N`
loop label ; `ecx=ecx-1` и если `ecx` не `0`
; переход на `label`
call quit
```

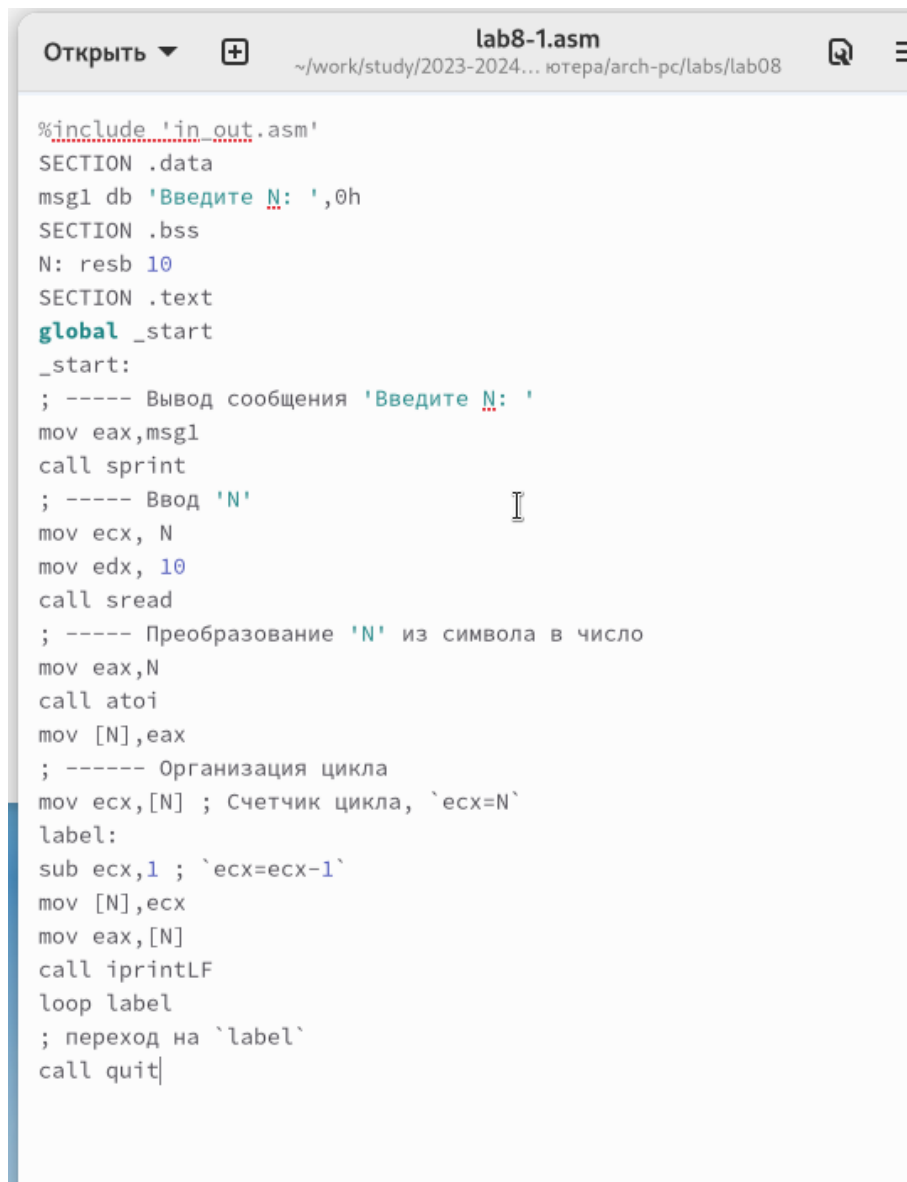
Рис. 4.1: Изменение кода lab8-1.asm

Затем создаю исполняемый файл и проверю его работу.(рис. [4.2])

```
[mdavletova@fedora lab08]$ nasm -f elf lab8-1.asm
[mdavletova@fedora lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1
[mdavletova@fedora lab08]$ ./lab8-1
Введите N: 4
4
3
2
1
[mdavletova@fedora lab08]$ ./lab8-1
Введите N: 3
3
2
1
[mdavletova@fedora lab08]$
```

Рис. 4.2: Компиляция текста программы lab8-1.asm

В данном примере демонстрируется, что использование регистра еsx в инструкции loop может привести к неправильной работе программы. В тексте программы вношу изменения, которые включают изменение значения регистра еsx внутри цикла.(рис. [4.3])



```
Открыть ▾ + lab8-1.asm
~/work/study/2023-2024... ютепа/arch-pc/labs/lab08

%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:
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF
loop label
; переход на `label`
call quit
```

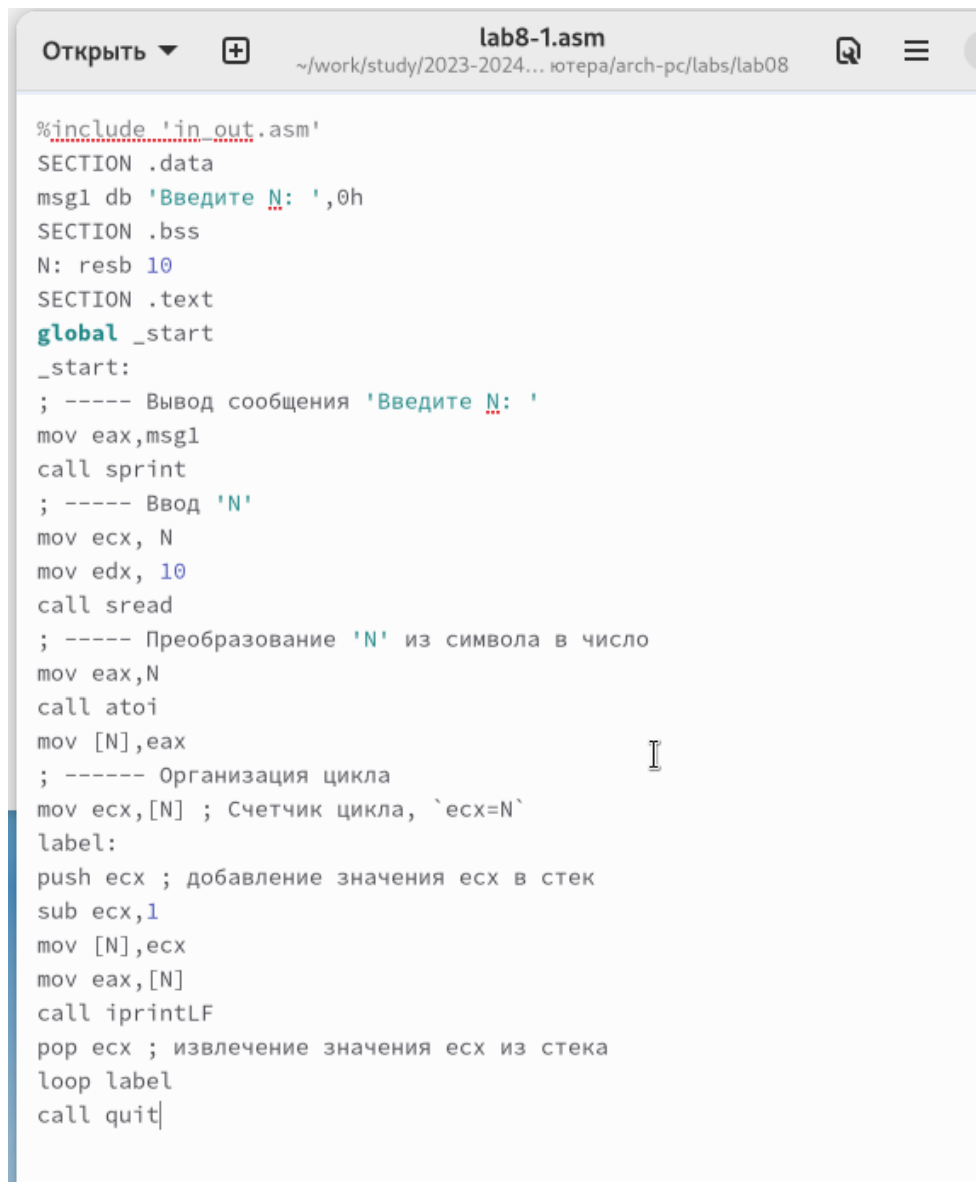
Рис. 4.3: Изменение кода lab8-1.asm

Программа запускает бесконечный цикл при нечетном значении N и выводит только нечетные числа при четном значении N. (рис. [4.4])

```
4294928934
4294928932
4294928930
4294928928
4294928926
4294928924
^C
[mdavletova@fedora lab08]$ ./lab8-1
Введите N: 4
3
1
[mdavletova@fedora lab08]$
```

Рис. 4.4: Компиляция текста программы lab8-1.asm

Для того чтобы использовать регистр `ecx` в цикле и обеспечить корректность работы программы, применяется стек. Вношу изменения в текст программы, добавив команды `push` и `pop` для сохранения значения счётчика цикла `loop` в стеке.(рис. [4.5])



```
Открыть ▾ + lab8-1.asm
~/work/study/2023-2024... ютеpa/arch-pc/labs/lab08

%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:
push ecx ; добавление значения ecx в стек
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx ; извлечение значения ecx из стека
loop label
call quit
```

Рис. 4.5: Изменение кода lab8-1.asm

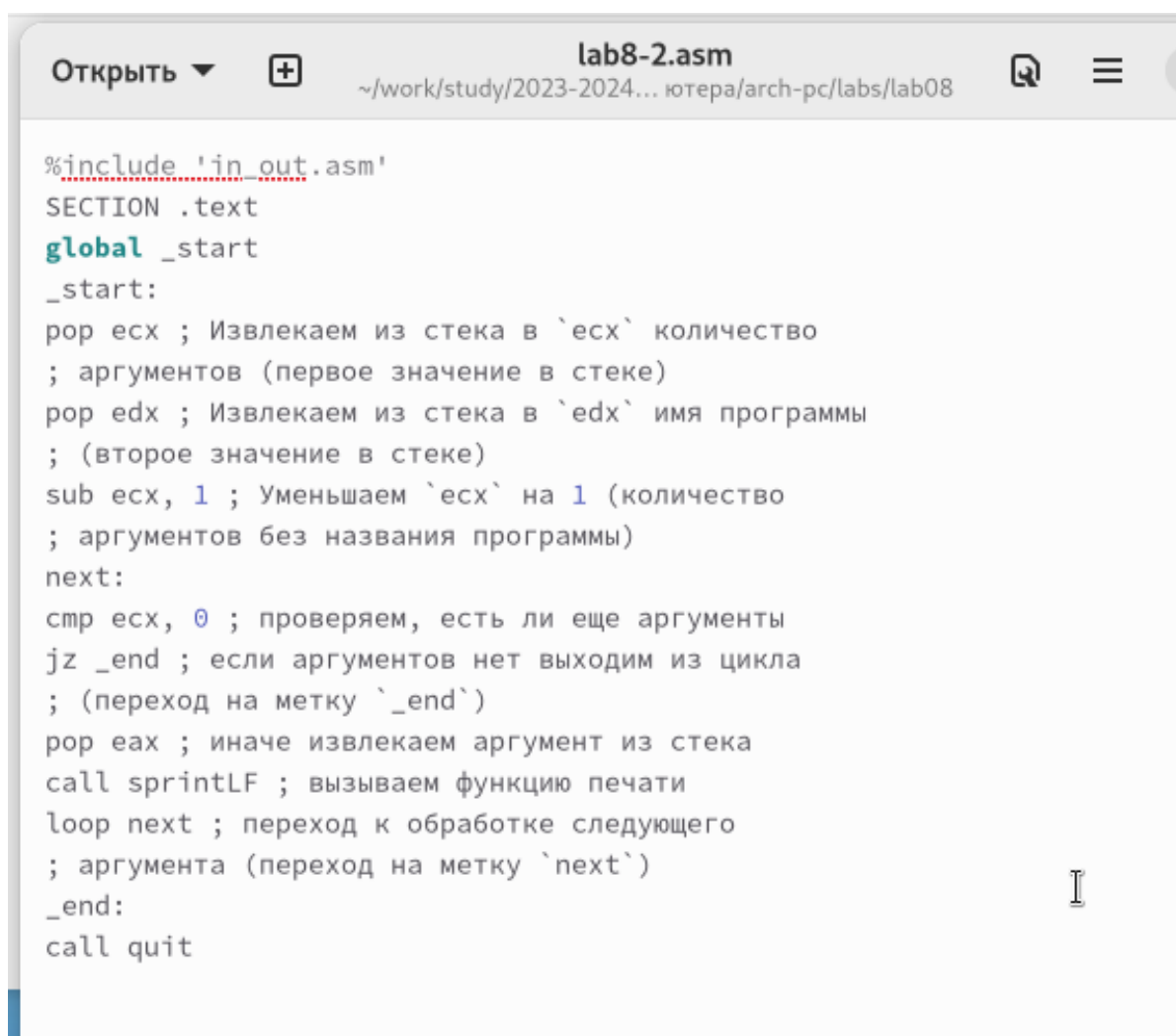
Был создан исполняемый файл и проверена его работа. Программа выводит числа от N-1 до 0, где количество проходов цикла соответствует значению N.(рис. [4.6])

```
[mdavletova@fedora lab08]$ nasm -f elf lab8-1.asm
[mdavletova@fedora lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1
[mdavletova@fedora lab08]$ ./lab8-1
Введите N: 3
2
1
0
[mdavletova@fedora lab08]$ ./lab8-1
Введите N: 4
3
2
1
0
[mdavletova@fedora lab08]$
```

Рис. 4.6: Компиляция текста программы lab8-1.asm

4.2 Обработка аргументов командной строки

Я создала файл lab8-2.asm в каталоге ~/work/arch-рс/lab08 и ввела в него текст программы из листинга 8.2. (рис. [4.7])



```
Открыть ▾ + lab8-2.asm
~/work/study/2023-2024... ютера/arch-pc/labs/lab08

%include 'in_out.asm'
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
next:
cmp ecx, 0 ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем аргумент из стека
call sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

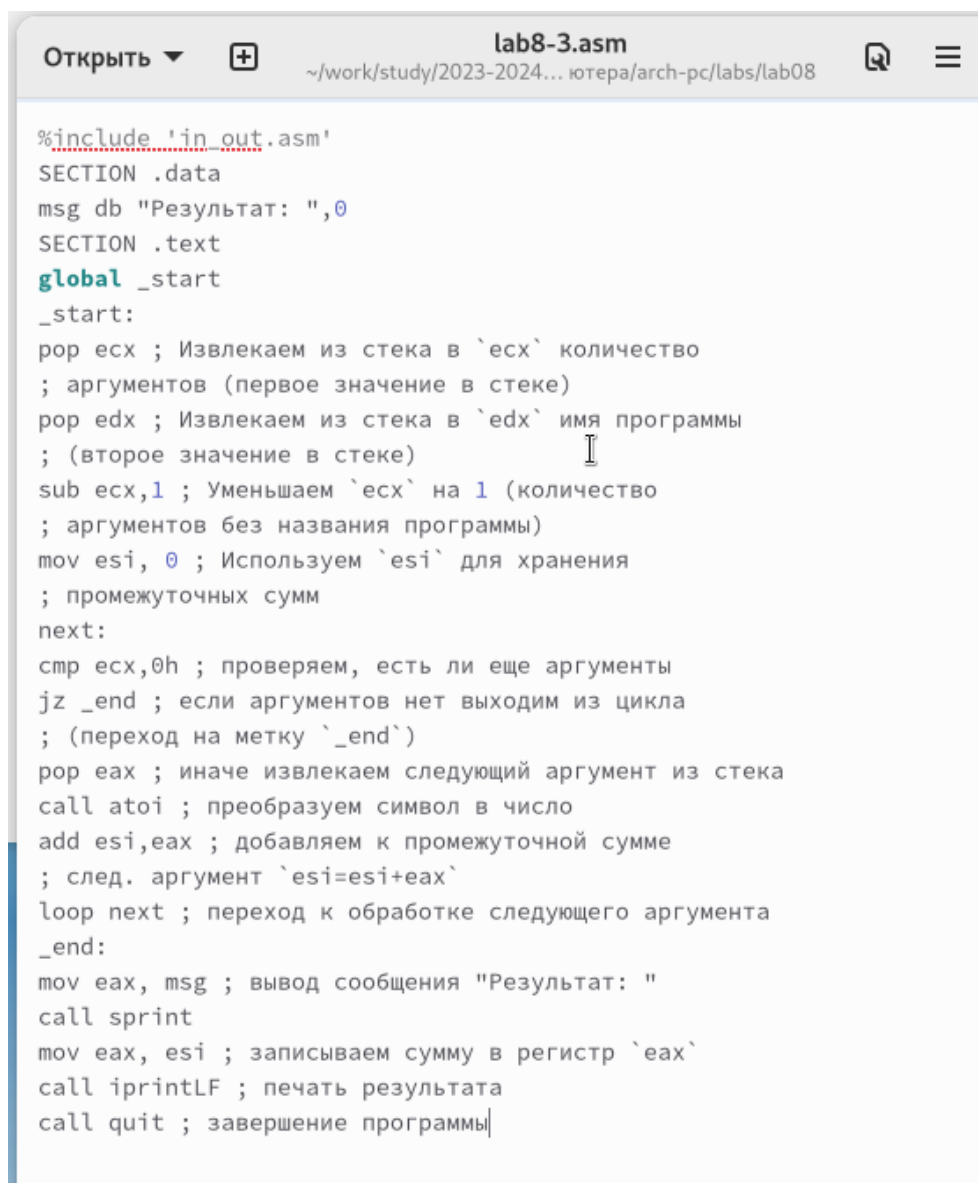
Рис. 4.7: Изменение кода lab8-2.asm




Затем я создала исполняемый файл и запустила его, указав аргументы. Программа успешно обработала 5 аргументов. Аргументами считаются слова/числа, разделенные пробелом. (рис. [4.8])

```
[mdavletova@fedora lab08]$  
[mdavletova@fedora lab08]$ nasm -f elf lab8-2.asm  
[mdavletova@fedora lab08]$ ld -m elf_i386 lab8-2.o -o lab8-2  
[mdavletova@fedora lab08]$ ./lab8-2  
[mdavletova@fedora lab08]$ ./lab8-2 argument 1 argument 2 'argument 3'  
argument  
1  
argument  
2  
argument 3  
[mdavletova@fedora lab08]$
```

Рис. 4.8: Компиляция текста программы lab8-2.asm

Теперь рассмотрим ещё один пример программы, которая выводит сумму чисел, переданных в программу как аргументы командной строки. (рис. [4.9]) (рис. [4.10])



```
Открыть ▾  lab8-3.asm
~/work/study/2023-2024... ютера/arch-pc/labs/lab08  

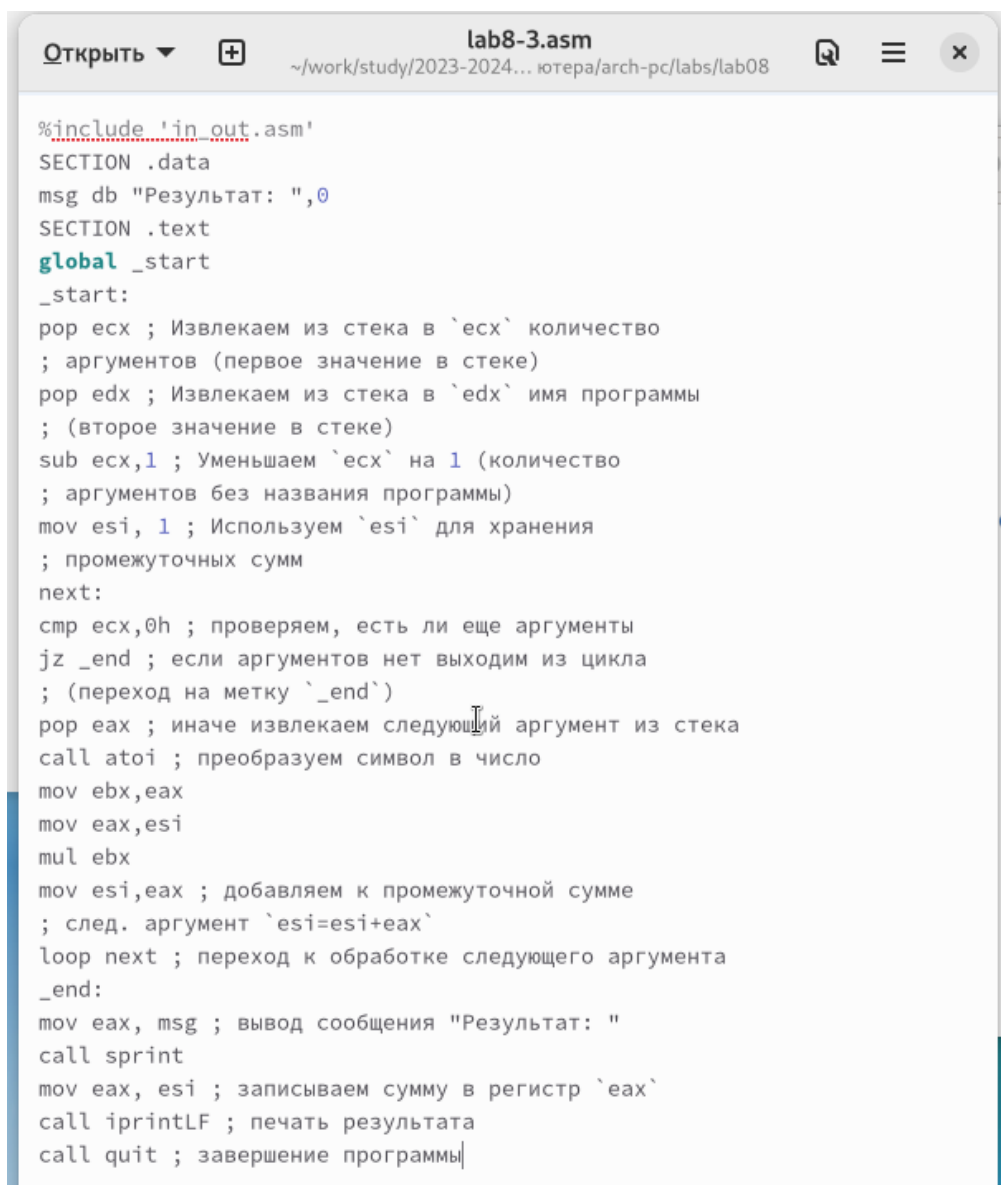
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 4.9: Изменение кода lab8-3.asm

```
[mdavletova@fedora lab08]$  
[mdavletova@fedora lab08]$ nasm -f elf lab8-3.asm  
[mdavletova@fedora lab08]$ ld -m elf_i386 lab8-3.o -o lab8-3  
[mdavletova@fedora lab08]$ ./lab8-3 3 4 5  
Результат: 12  
[mdavletova@fedora lab08]$  
[mdavletova@fedora lab08]$
```

Рис. 4.10: Компиляция текста программы lab8-3.asm

Я изменила текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис. [4.11]) (рис. [4.12])



```
lab8-3.asm
~/work/study/2023-2024... ютеpa/arch-pc/labs/lab08

%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 1 ; Используем `esi` для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
mov ebx,eax
mov eax,esi
mul ebx
mov esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 4.11: Изменение кода lab8-3.asm

```
[mdavletova@fedora lab08]$  
[mdavletova@fedora lab08]$ nasm -f elf lab8-3.asm  
[mdavletova@fedora lab08]$ ld -m elf_i386 lab8-3.o -o lab8-3  
[mdavletova@fedora lab08]$ ./lab8-3 3 4 5  
Результат: 60  
[mdavletova@fedora lab08]$
```

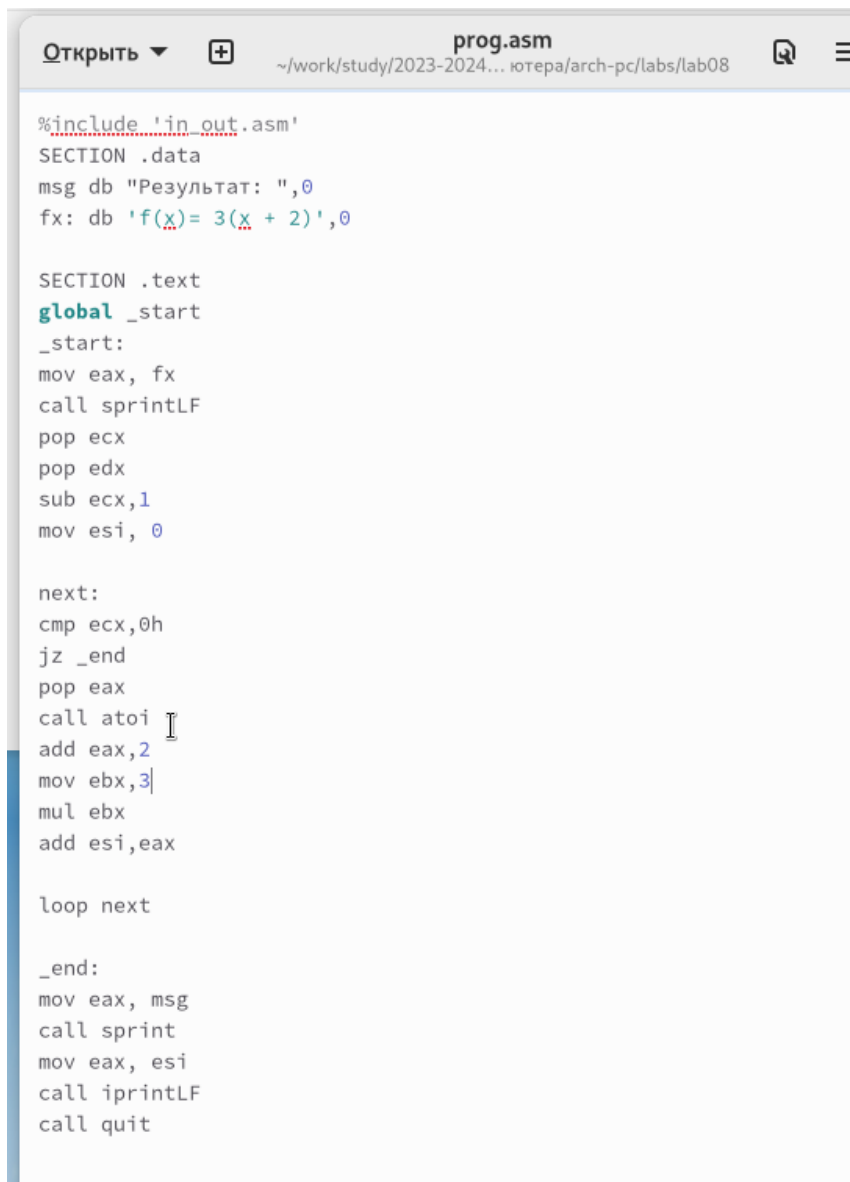
Рис. 4.12: Компиляция текста программы lab8-3.asm

4.3 Выполнение заданий для самостоятельной работы

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 передаются как аргументы. Вид функции $f(x)$ выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах x . (рис. [4.13]) (рис. [4.14])

Мой вариант 7:

$$f(x) = 3(x + 2)$$



```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
fx: db 'f(x)= 3(x + 2)',0

SECTION .text
global _start
_start:
mov eax, fx
call sprintLF
pop ecx
pop edx
sub ecx,1
mov esi, 0

next:
cmp ecx,0h
jz _end
pop eax
call atoi
add eax,2
mov ebx,3
mul ebx
add esi,eax

loop next

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

Рис. 4.13: Изменение кода prog.asm

```
[mdavletova@fedora lab08]$  
[mdavletova@fedora lab08]$ nasm -f elf prog.asm  
[mdavletova@fedora lab08]$ ld -m elf_i386 prog.o -o prog  
[mdavletova@fedora lab08]$ ./prog  
f(x)= 3(x + 2)  
Результат: 0  
[mdavletova@fedora lab08]$ 1  
bash: 1: команда не найдена...  
[mdavletova@fedora lab08]$ ./prog 1  
f(x)= 3(x + 2)  
Результат: 9  
[mdavletova@fedora lab08]$ ./prog 3 4 5  
f(x)= 3(x + 2)  
Результат: 54  
[mdavletova@fedora lab08]$
```

Рис. 4.14: Компиляция текста программы prog.asm

5 Выводы

Освоили работы со стеком, циклом и аргументами на ассемблере `naasm`.