

Отчёт по лабораторной работе № 8

**дисциплина: Архитектура компьютера. Программирование
цикла. Обработка аргументов командной строки**

Студент: Святашова Ксения Евгеньевна

Содержание

1	Цель работы	4
2	Теоритическое введение	5
3	Выполнение лабораторной работы	7
3.1	Реализация циклов в NASM	7
3.2	Обработка аргументов командной строки	13
4	Задания для самостоятельной работы	19
5	Вывод	23

Список иллюстраций

3.1	Каталог lab08	7
3.2	Ввод текста из листинга 8.1	8
3.3	Запуск исполняемого файла	10
3.4	Изменение текста программы	10
3.5	Запуск исполняемого файла	11
3.6	Изменение текста программы	12
3.7	Запуск исполняемого файла	12
3.8	Создание файла lab8-2.asm	13
3.9	Ввод текста из листинга 7.3	14
3.10	Запуск исполняемого файла	15
3.11	Создание файла lab8-3.asm	15
3.12	Ввод текста из листинга 8.3	16
3.13	Запуск исполняемого файла	17
3.14	Программа для вычисления произведения	18
3.15	Запуск исполняемого файла	18
4.1	Создание файла lab8-4.asm	19
4.2	Текст программы для варианта №9	20
4.3	Проверка программы	22
4.4	Проверка программы	22

1 Цель работы

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

2 Теоритическое 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.

Для стека существует две основные операции:

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

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.

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.

Аналогично команде записи в стек существует команда рора, которая восстанавливает из стека все регистры общего назначения, и команда рорf для перемещения значений из вершины стека в регистр флагов.

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре есх. Наиболее простой является инструкция loор. Она позволяет организовать безусловный цикл.

Иструкция loор выполняется в два этапа. Сначала из регистра есх 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 loор.

3 Выполнение лабораторной работы

3.1 Реализация циклов в NASM

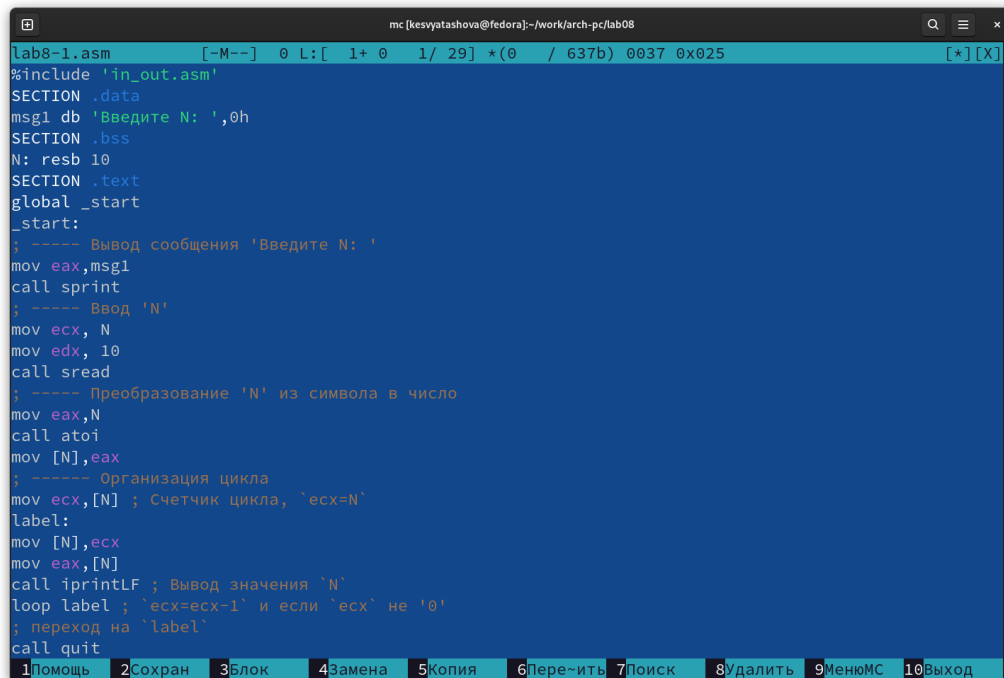
Создадим каталог для программ лабораторной работы №8, перейдем в него и создадим файл lab8-1.asm(рис. 3.1):

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

Рис. 3.1: Каталог lab08

При реализации циклов в NASM с использованием инструкции `loop` необходимо помнить о том, что эта инструкция использует регистр `ecx` в качестве счетчика и на каждом шаге уменьшает его значение на единицу.

Введем в файл `lab8-1.asm` текст программы из листинга 8.1.(рис. 3.2):



```
lab8-1.asm      [-M--]  0 L:[ 1+ 0 1/ 29] *(0 / 637b) 0037 0x025      [*][X]
#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
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пере-ить 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 3.2: Ввод текста из листинга 8.1

Листинг 8.1. 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]
call
iprintLF ; Вывод значения N
loop label
call
quit

```

Создадим исполняемый файл и проверим его работу(рис. 3.3):

```

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

```

Рис. 3.3: Запуск исполняемого файла

Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы.

Изменим текст программы(рис. 3.4), добавив изменение значение регистра `ecx` в цикле:

```

label:
sub ecx,1 ; 'ecx=ecx-1
mov [N],ecx
mov eax,[N]
call iprintLF
loop label

```

```

label:
sub ecx,1 ; ecx=ecx-1
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
loop label ; `ecx=ecx-1` и если `ecx` не '0'

```

Рис. 3.4: Изменение текста программы

Создадим исполняемый файл и запустим его(рис. 3.5):

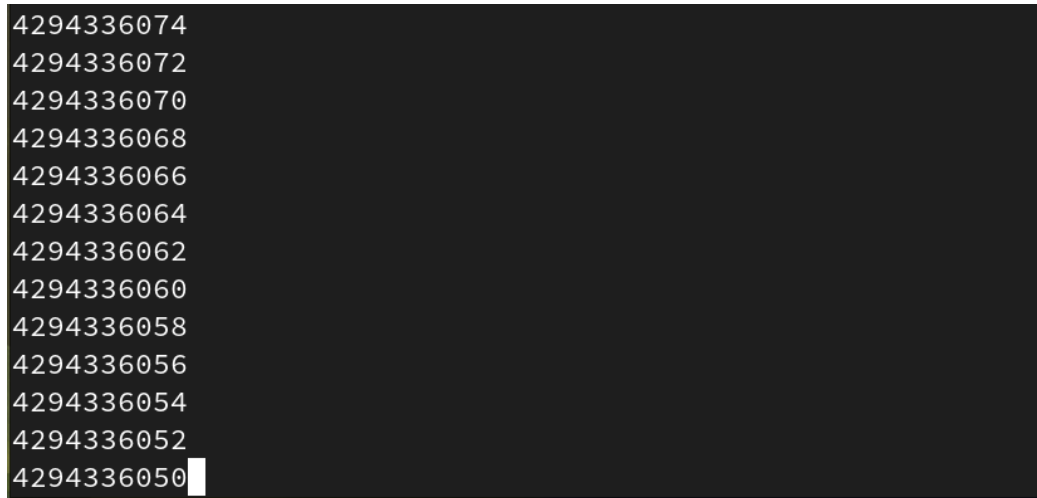


Рис. 3.5: Запуск исполняемого файла

При данном изменении программы цикл закольцевался и стал бесконечным.

Внесем изменения(рис. 3.6) в текст программы, добавив команды push и pop(добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop:

```
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
```

```

label:
push ecx
sub ecx,1 ; ecx=ecx-1
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
pop ecx
loop label ; `ecx=ecx-1` и если `ecx` не '0'

```

Рис. 3.6: Изменение текста программы

Создадим исполняемый файл и проверим его работу(рис. 3.7):

```

kesvyatashova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
kesvyatashova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
kesvyatashova@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
kesvyatashova@fedora:~/work/arch-pc/lab08$

```

Рис. 3.7: Запуск исполняемого файла

Теперь, после изменения программы, число циклов стало соответство-

вать числу, введенному с клавиатуры.

3.2 Обработка аргументов командной строки

При разработке программ иногда встает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы.

При запуске программы в NASM аргументы командной строки загружаются в стек в обратном порядке, кроме того в стек записывается имя программы и общее количество аргументов. Последние два элемента стека для программы, скомпилированной NASM,—это всегда имя программы и количество переданных аргументов.

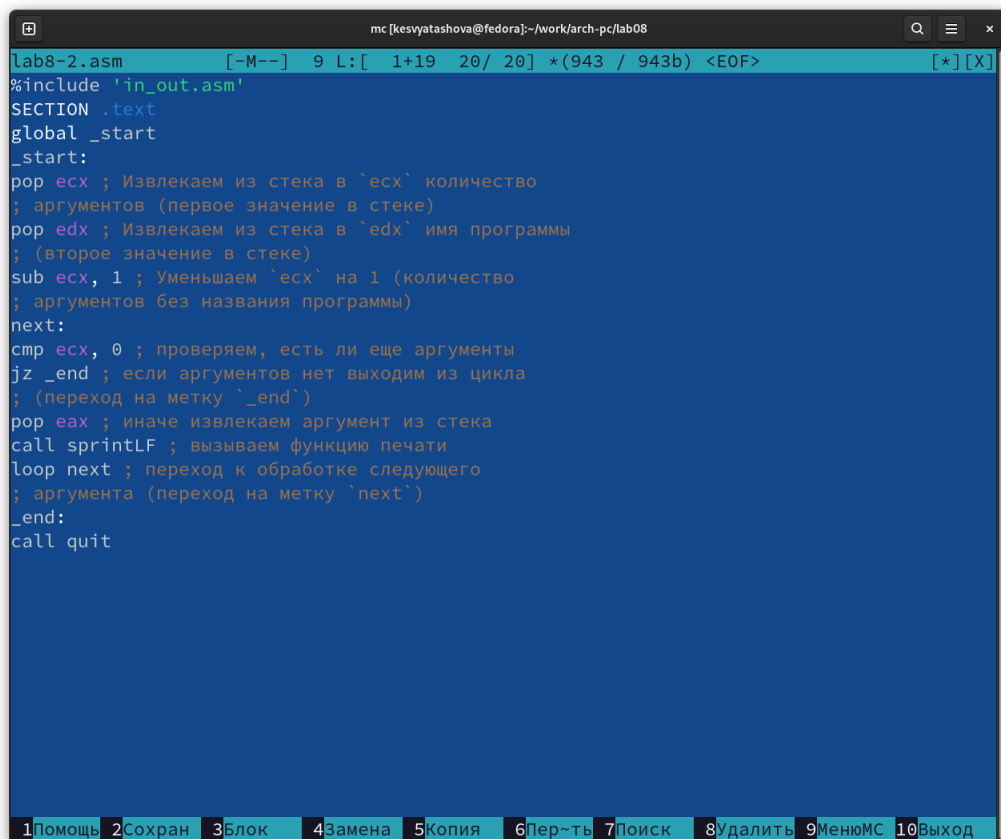
Таким образом, для того чтобы использовать аргументы в программе, их нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. Т.е. сначала нужно извлечь из стека количество аргументов, а затем циклично для каждого аргумента выполнить логику программы.

Создадим файл lab8-2.asm (рис. 3.8) в каталоге ~/work/arch-pc/lab08 и введем в него текст программы из листинга 8.2. (рис. 3.9):



```
kesvyatashova@fedora:~/work/arch-pc/lab08$ touch lab8-2.asm
kesvyatashova@fedora:~/work/arch-pc/lab08$
```

Рис. 3.8: Создание файла lab8-2.asm



```
lab8-2.asm      [-M--]  9 L:[ 1+19 20/ 20] *(943 / 943b) <EOF>      [*] [X]
#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 sprintf ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

Рис. 3.9: Ввод текста из листинга 7.3

Листинг 8.2. Программа выводящая на экран аргументы командной строки

```
%include 'in_out.asm'

SECTION .text

global _start

_start:

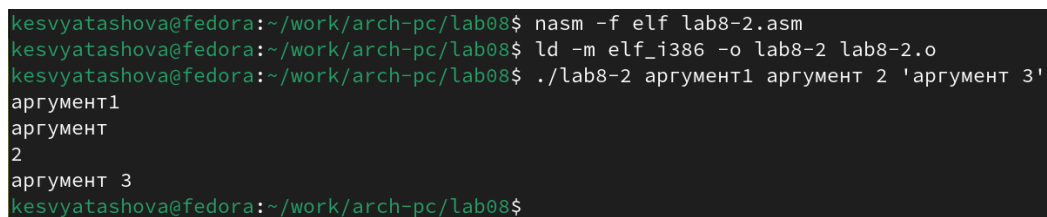
pop ecx

pop edx

sub ecx, 1
```

```
next:
cmp ecx, 0
jz _end
pop eax
call sprintLF
loop next
_end:
call quit
```

Создадим исполняемый файл и запустим его, указав аргументы(рис. 3.10):

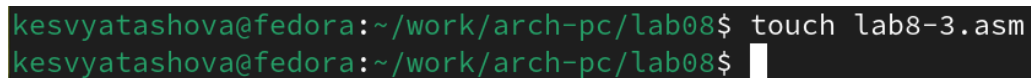


```
kesvyatashova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
kesvyatashova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
kesvyatashova@fedora:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
kesvyatashova@fedora:~/work/arch-pc/lab08$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

Рис. 3.10: Запуск исполняемого файла

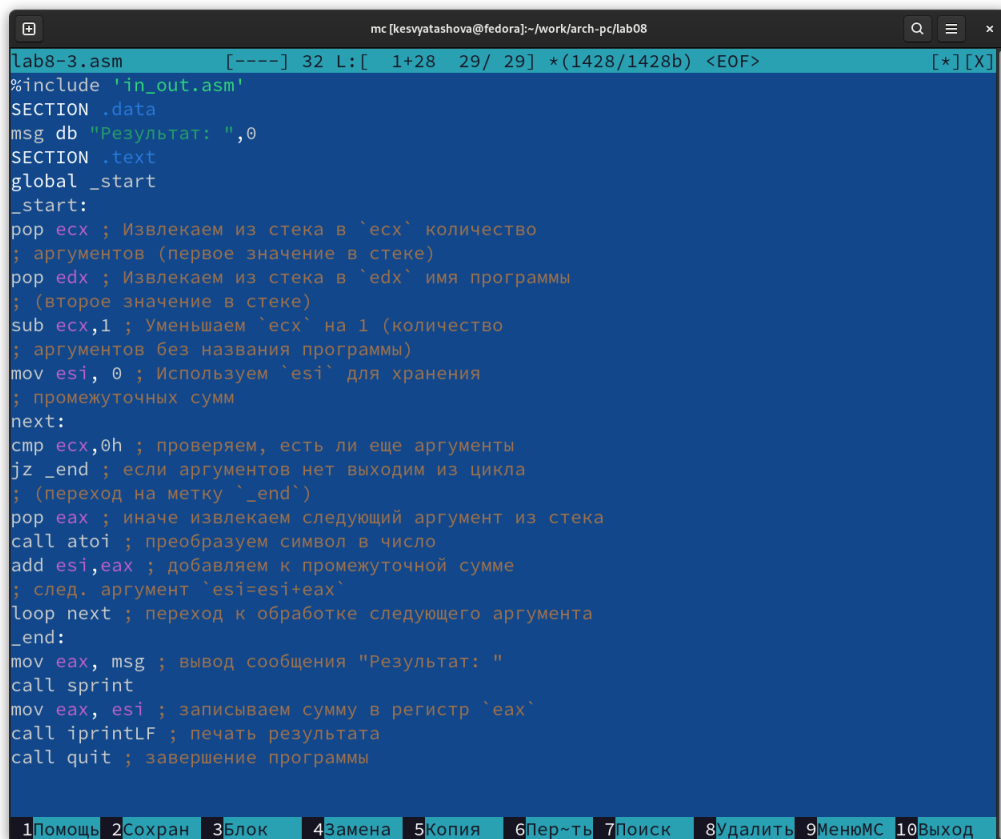
Программой было обработано 4 аргумента.

Создадим файл lab8-3.asm(рис. 3.11) в каталоге ~/work/archpc/lab08 и введем в него текст программы из листинга 8.3.(рис. 3.12):



```
kesvyatashova@fedora:~/work/arch-pc/lab08$ touch lab8-3.asm
kesvyatashova@fedora:~/work/arch-pc/lab08$
```

Рис. 3.11: Создание файла lab8-3.asm



```
lab8-3.asm      [----] 32 L: [ 1+28 29/ 29] *(1428/1428b) <EOF>  [*] [X]
#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

1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 3.12: Ввод текста из листинга 8.3

Листинг 8.3. Программа вычисления суммы аргументов командной строки

```
%include 'in_out.asm'

SECTION .data
msg db "Результат:",0

SECTION .text
global _start

_start:
pop ecx
```

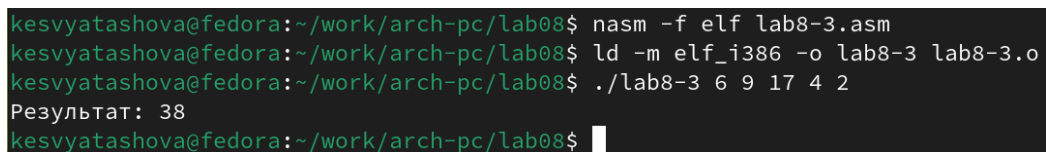


```

pop edx
sub ecx,1
mov esi, 0
next:
cmp ecx,0h
jz _end
pop eax
call atoi
add esi,eax
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

```

Создадим исполняемый файл и запустим его, указав аргументы(рис. 3.13):



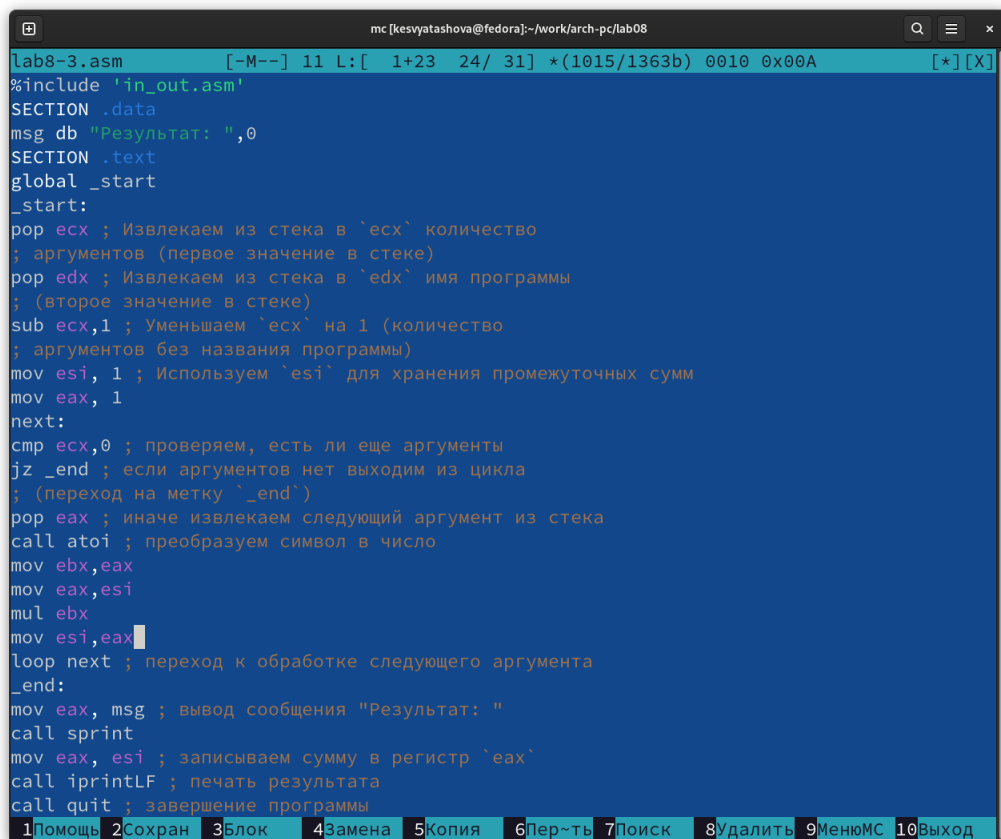
```

kesvyatashova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
kesvyatashova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
kesvyatashova@fedora:~/work/arch-pc/lab08$ ./lab8-3 6 9 17 4 2
Результат: 38
kesvyatashova@fedora:~/work/arch-pc/lab08$

```

Рис. 3.13: Запуск исполняемого файла

Изменим текст программы из листинга 8.3 для вычисления произведения аргументов командной строки(рис. 3.14):



```
lab8-3.asm [-M--] 11 L: [ 1+23 24/ 31] *(1015/1363b) 0010 0x00A [*] [X]
#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
mov eax, 1
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 следующий аргумент из стека
call atoi ; преобразуем символ в число
mov ebx,eax
mov eax,esi
mul ebx
mov 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
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 3.14: Программа для вычисления произведения

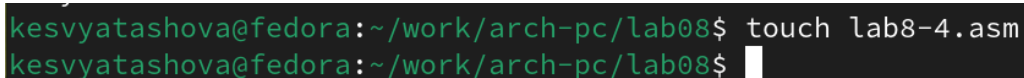
Создадим исполняемый файл и запустим его, указав аргументы(рис. 3.15):

```
kesvyatashova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
kesvyatashova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
kesvyatashova@fedora:~/work/arch-pc/lab08$ ./lab8-3 5 8 20 11 1
Результат: 8800
kesvyatashova@fedora:~/work/arch-pc/lab08$
```

Рис. 3.15: Запуск исполняемого файла

4 Задания для самостоятельной работы

1. Создадим файл для выполнения самостоятельной работы(рис. 4.1):



```
kesvyatashova@fedora:~/work/arch-pc/lab08$ touch lab8-4.asm
kesvyatashova@fedora:~/work/arch-pc/lab08$
```

Рис. 4.1: Создание файла lab8-4.asm

Напишем программу(рис. 4.2), 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 $f(x)$ я выбрала в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Мой вариант-9, значит моя функция: $10x-4$.

```
lab8-4.asm  [----] 10 L: [ 1+28 29/ 29] *(342 / 342b) <EOF> [*] [X]
#include 'in_out.asm'
SECTION .data
prim db 'f(x)=10x-4',0
otv db 'Ответ: ',0
SECTION .text
GLOBAL _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi,0
mov eax,prim
call sprintLF
next:
cmp ecx,0
jz _end
mov ebx,10
pop eax
call atoi
mul ebx
add eax,-4
add esi,eax
loop next
_end:
mov eax,otv
call sprint
mov eax,esi
call iprintLF
call quit
```

1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Удалить 9МенюМС 10Выход

Рис. 4.2: Текст программы для варианта №9

Текст программы:

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
prim db 'f(x)=10x-4',0
```

```
otv db 'Ответ:',0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
pop ecx
```

```

pop edx
sub ecx,1
mov esi,0
mov eax,prim
call sprintLF
next:
cmp ecx,0
jz _end
mov ebx,10
pop eax
call atoi
mul ebx
add eax,-4
add esi,eax
loop next
_end:
mov eax,otv
call sprint
mov eax,esi
call iprintLF
call quit

```

Создадим исполняемый файл и проверим его работу на нескольких наборах $x=x_1, x_2, \dots, x_n$. (рис. 4.3) и (рис. 4.4):

```
kesvyatashova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-4.asm
kesvyatashova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-4 lab8-4.o
kesvyatashova@fedora:~/work/arch-pc/lab08$ ./lab8-4 7 2 13 10
f(x)=10x-4
Ответ: 304
kesvyatashova@fedora:~/work/arch-pc/lab08$
```

Рис. 4.3: Проверка программы

```
kesvyatashova@fedora:~/work/arch-pc/lab08$ ./lab8-4 19 4 16 34 5
f(x)=10x-4
Ответ: 760
kesvyatashova@fedora:~/work/arch-pc/lab08$
```

Рис. 4.4: Проверка программы

5 Вывод

В результате выполнения работы я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки.