

Отчёта по лабораторной работе 9

Программирование цикла. Обработка аргументов командной строки.

Нефедова Наталия Николаевна НПИбд-02-22

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Выводы	21
	Список литературы	22

Список иллюстраций

4.1	Файл lab9-1.asm	9
4.2	Работа программы lab9-1.asm	10
4.3	Файл lab9-1.asm	11
4.4	Работа программы lab9-1.asm	12
4.5	Файл lab9-1.asm	13
4.6	Работа программы lab9-1.asm	14
4.7	Файл lab9-2.asm	15
4.8	Работа программы lab9-2.asm	15
4.9	Файл lab9-3.asm	16
4.10	Работа программы lab9-3.asm	16
4.11	Файл lab9-3.asm	17
4.12	Работа программы lab9-3.asm	18
4.13	Файл lab9-4.asm	19
4.14	Работа программы lab9-4.asm	20

Список таблиц

1 Цель работы

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

2 Задание

1. Изучите примеры программ
2. 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 9.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах x .
3. Загрузите файлы на GitHub.

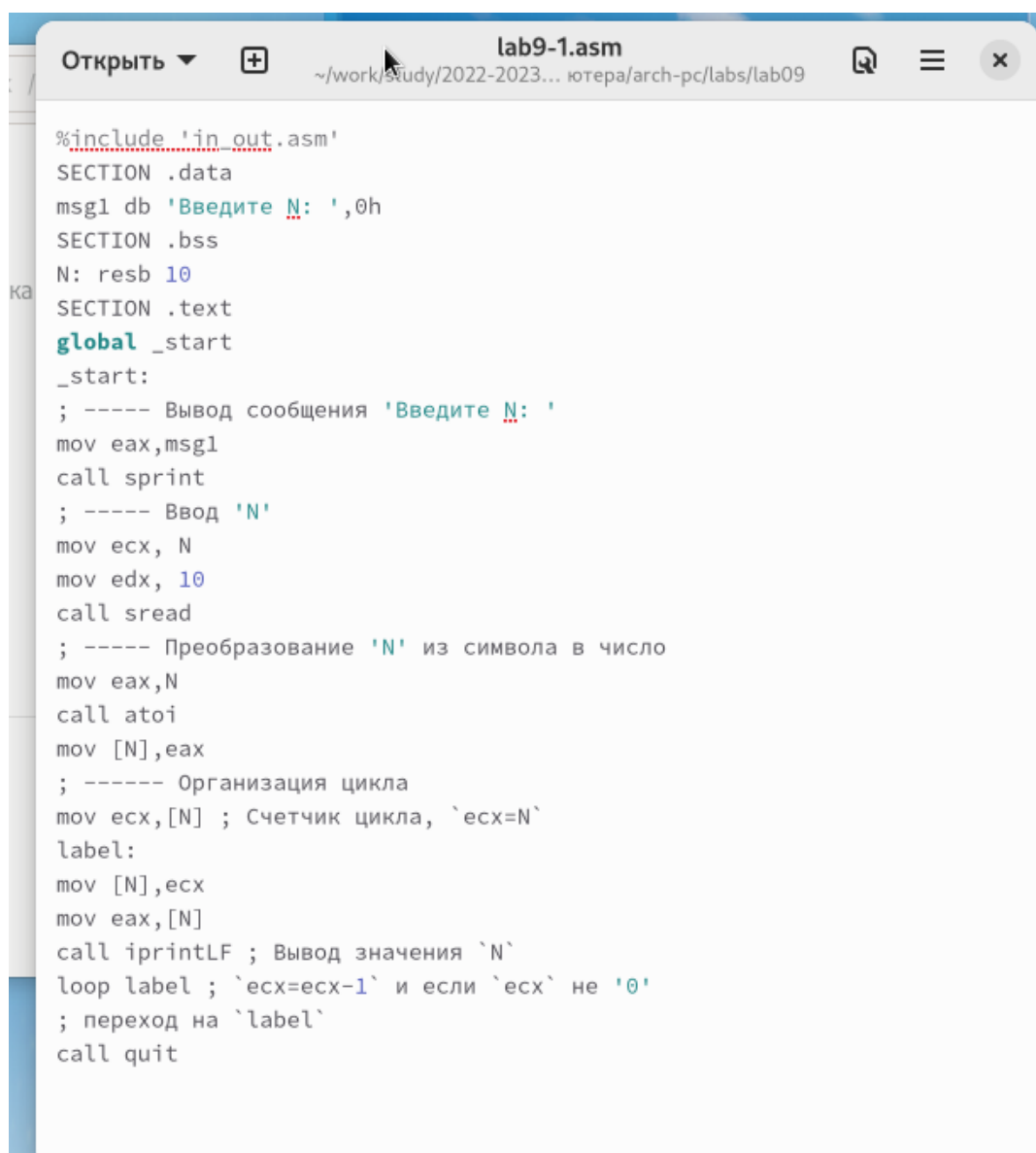
3 Теоретическое введение

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.

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре esx. Наиболее простой является инструкция loor. Инструкция loor выполняется в два этапа. Сначала из регистра esx 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 loor.

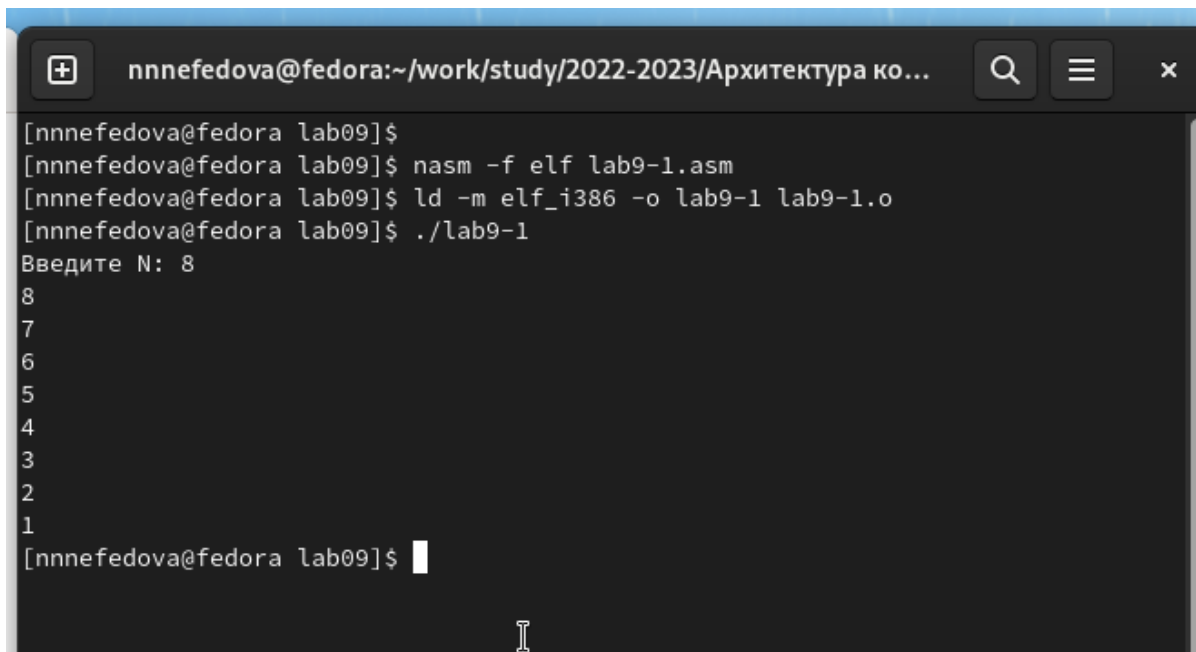
4 Выполнение лабораторной работы

1. Создайте каталог для программ лабораторной работы № 9, перейдите в него и создайте файл lab9-1.asm
2. Введите в файл lab9-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу. (рис. 4.1, 4.2)



```
%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: Файл lab9-1.asm



```
nnnefedova@fedora:~/work/study/2022-2023/Архитектура ко...
[nnnefedova@fedora lab09]$
[nnnefedova@fedora lab09]$ nasm -f elf lab9-1.asm
[nnnefedova@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[nnnefedova@fedora lab09]$ ./lab9-1
Введите N: 8
8
7
6
5
4
3
2
1
[nnnefedova@fedora lab09]$
```

Рис. 4.2: Работа программы lab9-1.asm

3. Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. Измените текст программы добавив изменение значение регистра `ecx` в цикле: Создайте исполняемый файл и проверьте его работу. Какие значения принимает регистр `ecx` в цикле? Соответствует ли число проходов цикла значению `N`, введенному с клавиатуры? (рис. 4.3, 4.4)

Программа запускает бесконечный цикл при нечетном `N` и выводит только нечетные числа при четном `N`.

```
%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
```

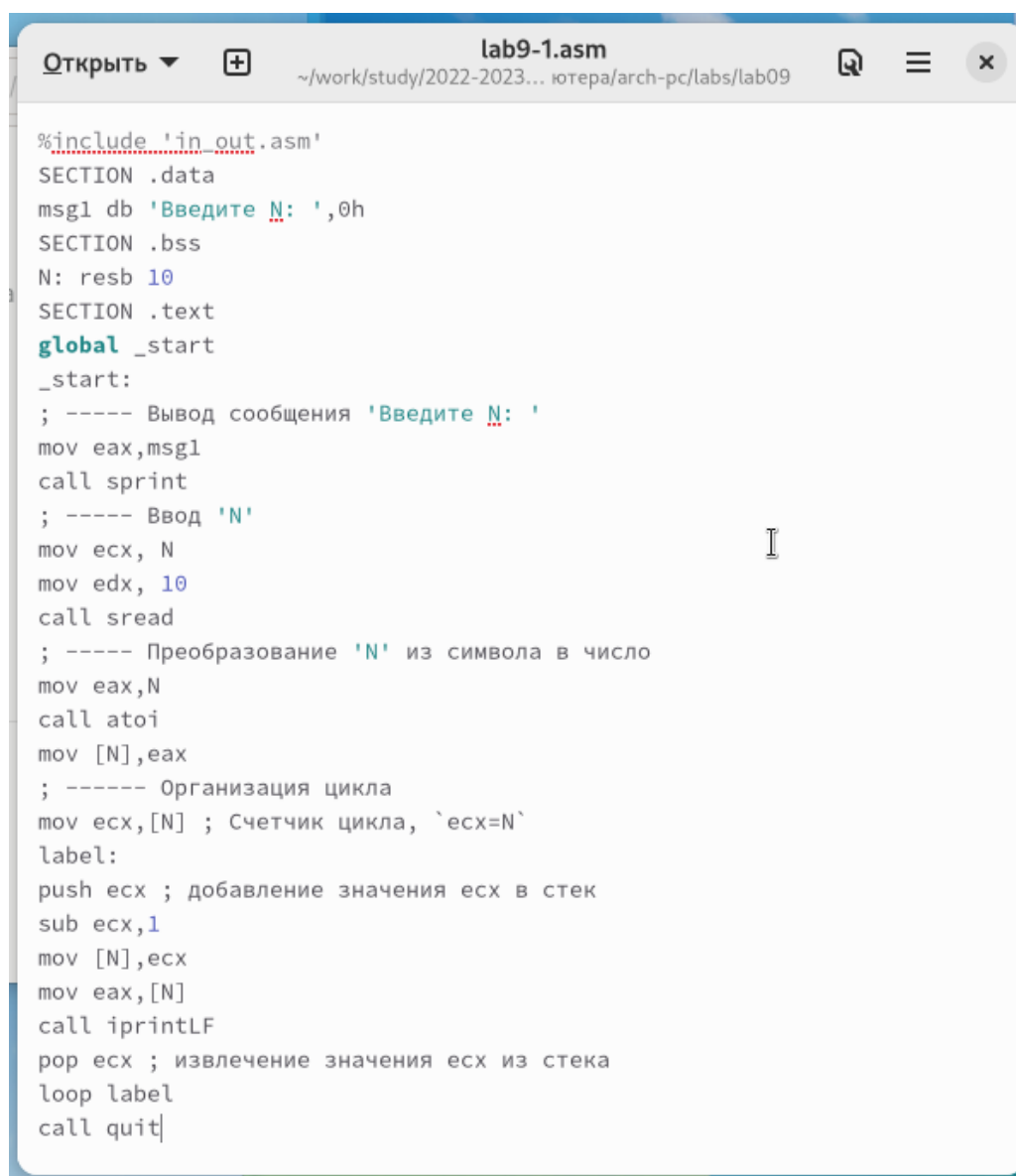
Рис. 4.3: Файл lab9-1.asm

```
4294917776
4294917774
4294917772
4294917770
4294917768
4294917766
4294917764
4294917762
4294917760
4294917758
42^C
[nnnefedova@fedora lab09]$ ./lab9-1
Введите N: 8
7
5
3
1
[nnnefedova@fedora lab09]$
```

Рис. 4.4: Работа программы lab9-1.asm

4. Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Внесите изменения в текст программы добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop`. Создайте исполняемый файл и проверьте его работу. Соответствует ли в данном случае число проходов цикла значению `N` введенному с клавиатуры? (рис. 4.5, 4.6)

Программа выводит числа от `N-1` до `0`, число проходов цикла соответствует `N`.



```
%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
```

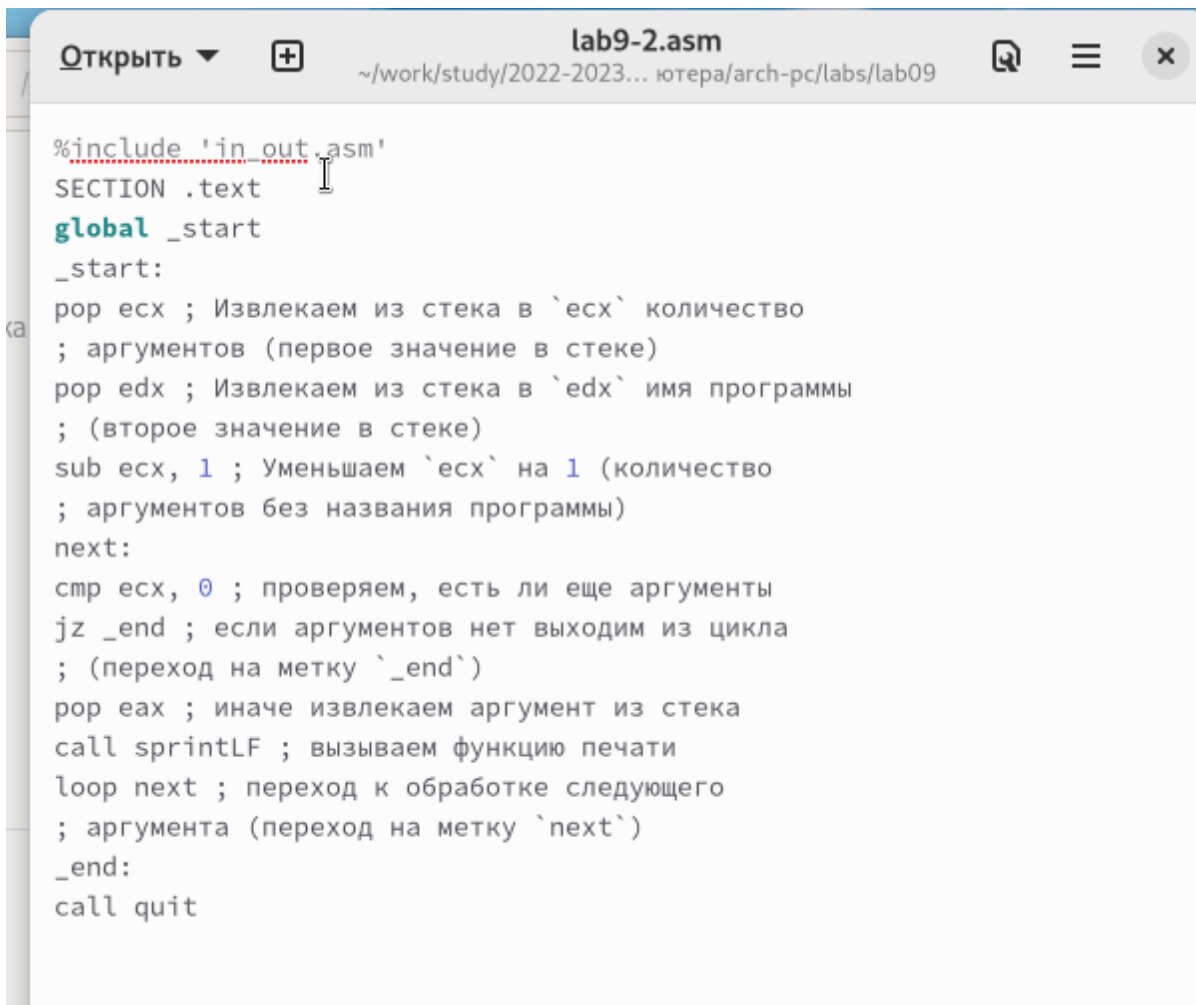
Рис. 4.5: Файл lab9-1.asm

```
[nnnefedova@fedora lab09]$ nasm -f elf lab9-1.asm
[nnnefedova@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[nnnefedova@fedora lab09]$ ./lab9-1
Введите N: 8
7
6
5
4
3
2
1
0
[nnnefedova@fedora lab09]$ ./lab9-1
Введите N: 3
2
1
0
[nnnefedova@fedora lab09]$
```

Рис. 4.6: Работа программы lab9-1.asm

5. Создайте файл lab9-2.asm в каталоге ~/work/arch-рс/lab09 и введите в него текст программы из листинга 9.2. Создайте исполняемый файл и запустите его, указав аргументы. (рис. 4.7, 4.8) Сколько аргументов было обработано программой?

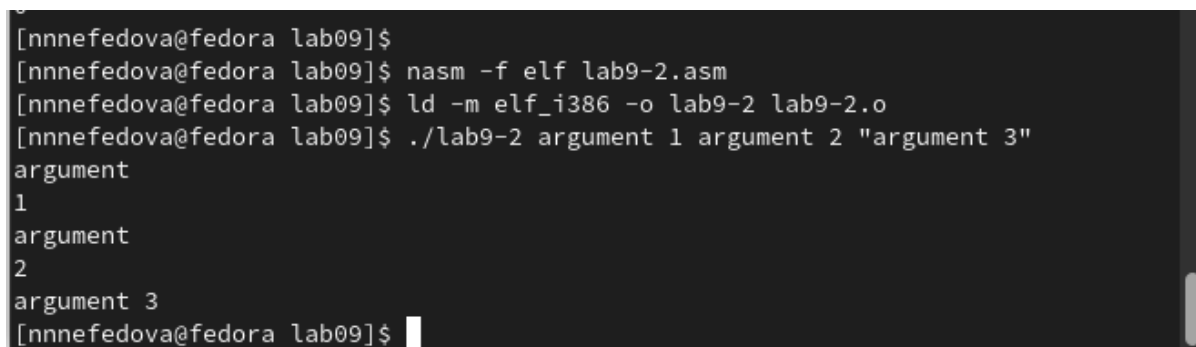
Программа обработала 5 аргументов.



```
Открыть ▾ + lab9-2.asm
~/work/study/2022-2023... ютеpa/arch-pc/labs/lab09

%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
```

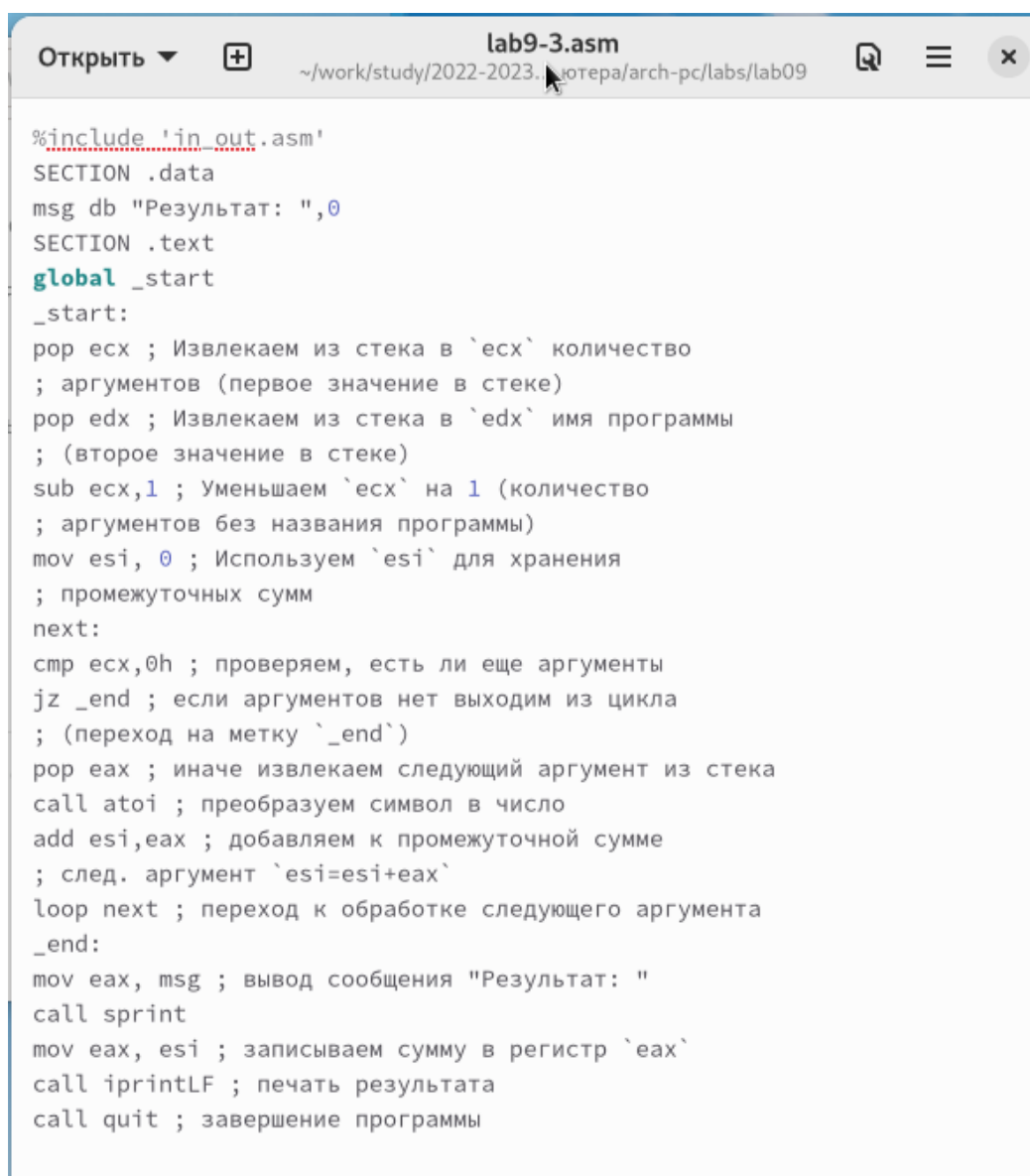
Рис. 4.7: Файл lab9-2.asm



```
[nnnefedova@fedora lab09]$
[nnnefedova@fedora lab09]$ nasm -f elf lab9-2.asm
[nnnefedova@fedora lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[nnnefedova@fedora lab09]$ ./lab9-2 argument 1 argument 2 "argument 3"
argument
1
argument
2
argument 3
[nnnefedova@fedora lab09]$
```

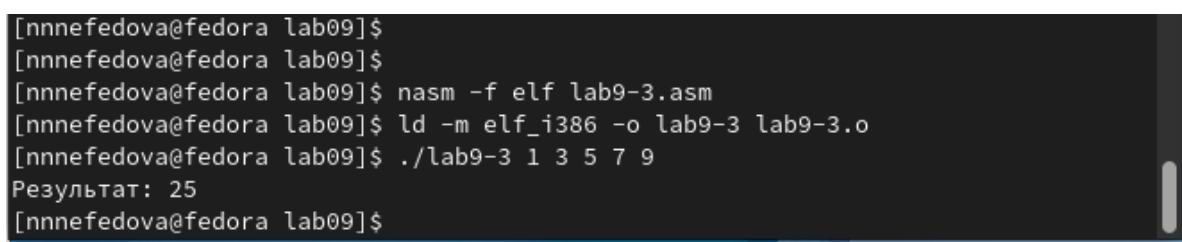
Рис. 4.8: Работа программы lab9-2.asm

6. Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. (рис. 4.9, 4.10)



```
%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: Файл lab9-3.asm

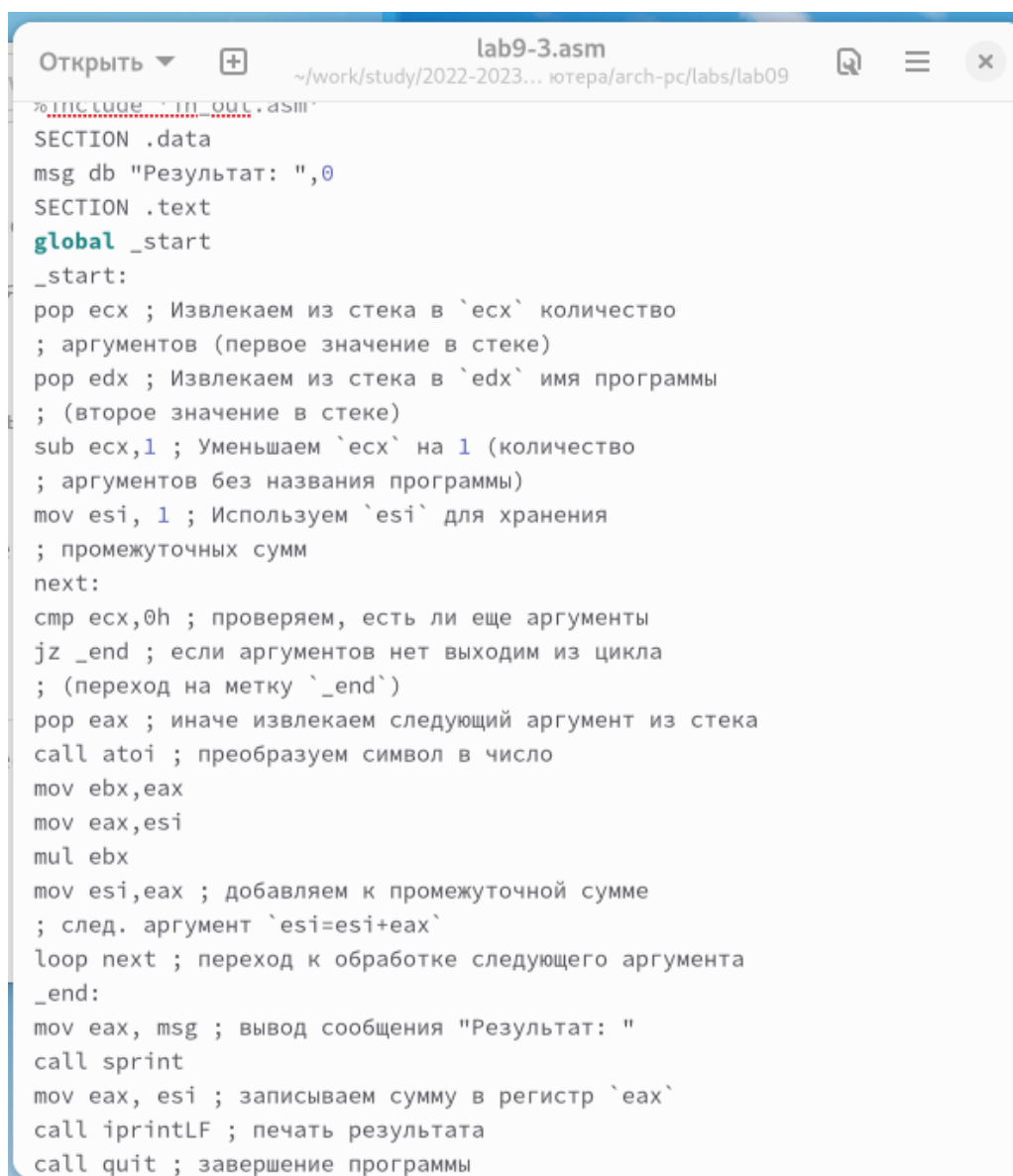


```
[nnnefedova@fedora lab09]$
[nnnefedova@fedora lab09]$
[nnnefedova@fedora lab09]$ nasm -f elf lab9-3.asm
[nnnefedova@fedora lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[nnnefedova@fedora lab09]$ ./lab9-3 1 3 5 7 9
Результат: 25
[nnnefedova@fedora lab09]$
```

Рис. 4.10: Работа программы lab9-3.asm

7. Измените текст программы из листинга 9.3 для вычисления произведения

аргументов командной строки. (рис. 4.11, 4.12)



```
Открыть ▾ + lab9-3.asm
~/work/study/2022-2023... ютеpa/arch-pc/labs/lab09
%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: Файл lab9-3.asm

```
argument 3
[nnnefedova@fedora lab09]$
[nnnefedova@fedora lab09]$
[nnnefedova@fedora lab09]$ nasm -f elf lab9-3.asm
[nnnefedova@fedora lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[nnnefedova@fedora lab09]$ ./lab9-3 1 3 5 7 9
Результат: 25
[nnnefedova@fedora lab09]$ nasm -f elf lab9-3.asm
[nnnefedova@fedora lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[nnnefedova@fedora lab09]$ ./lab9-3 1 3 5 7 9
Результат: 945
[nnnefedova@fedora lab09]$
```

Рис. 4.12: Работа программы lab9-3.asm

8. 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 9.1 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, 4.14)

для варианта 18 $f(x) = 17 + 5x$

```
Открыть ▾ + lab9-4.asm
~/work/study/2022-2023... ютера/arch-pc/labs/lab09

%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
fx: db 'f(x)=17+5x ',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

mov ebx,5
mul ebx
add eax,17
add esi,eax

loop next

_end:
mov eax, msg
call sprint
```

Рис. 4.13: Файл lab9-4.asm

```
[nnnefedova@fedora lab09]$  
[nnnefedova@fedora lab09]$ nasm -f elf lab9-4.asm  
[nnnefedova@fedora lab09]$ ld -m elf_i386 -o lab9-4 lab9-4.o  
[nnnefedova@fedora lab09]$ ./lab9-4 1 3 5 7 9  
f(x)=17+5x  
Результат: 210  
[nnnefedova@fedora lab09]$ ./lab9-4 2 4 6 8 10  
f(x)=17+5x  
Результат: 235  
[nnnefedova@fedora lab09]$
```

Рис. 4.14: Работа программы lab9-4.asm

5 Выводы

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

Список литературы

1. Расширенный ассемблер: NASM
2. MASM, TASM, FASM, NASM под Windows и Linux