

Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки.

Казначеев Сергей Ильич

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выполнение задания для самостоятельной работы	17
4	Выводы	19

Список иллюстраций

2.1	01	6
2.2	02	7
2.3	03	8
2.4	04	8
2.5	05	9
2.6	06	9
2.7	07	10
2.8	08	11
2.9	09	12
2.10	10	12
2.11	11	13
2.12	12	13
2.13	13	14
2.14	14	14
2.15	15	15
2.16	16	15
2.17	17	16
2.18	18	16
3.1	19	17
3.2	20	18
3.3	21	18

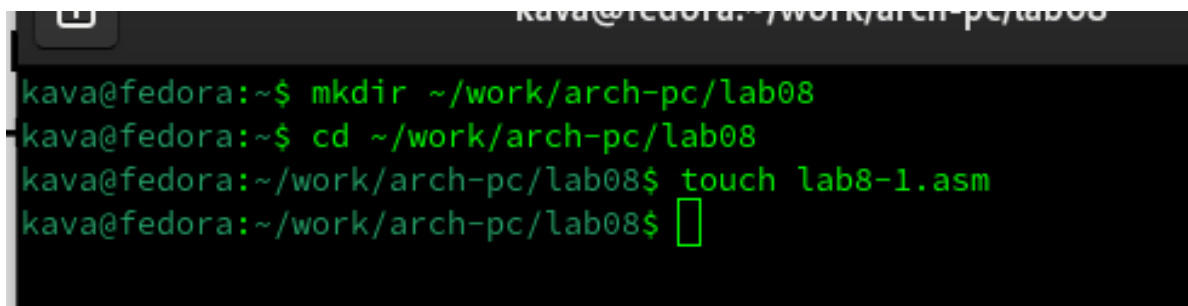
Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

Для начала создадим файл lab8-1.asm.

A terminal window with a dark background and green text. The prompt is 'kava@fedora:~'. The user enters 'mkdir ~/work/arch-pc/lab08', then 'cd ~/work/arch-pc/lab08', and finally 'touch lab8-1.asm'. The prompt changes to 'kava@fedora:~/work/arch-pc/lab08\$' after each command.

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

Рис. 2.1: 01

Далее запускаем Midnight commander через команду `mc` теперь вставляем в ранее созданный файл код из листинга 8.1. Он должен запускать цикл и выводить каждую итерацию числа , на единицу меньше предыдущего.

```
GNU nano 7.2 /home/kava/work/arch-pc/lab08/lab8
#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
```

Рис. 2.2: 02

Копируем файл in_out.asm, чтобы собирать файл.

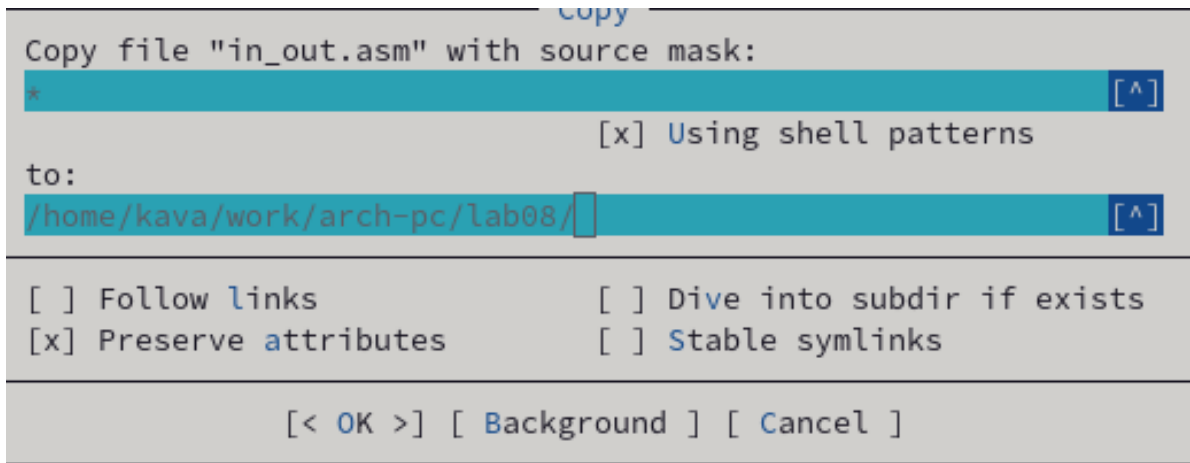


Рис. 2.3: 03

Теперь собираем программу и запускаем.

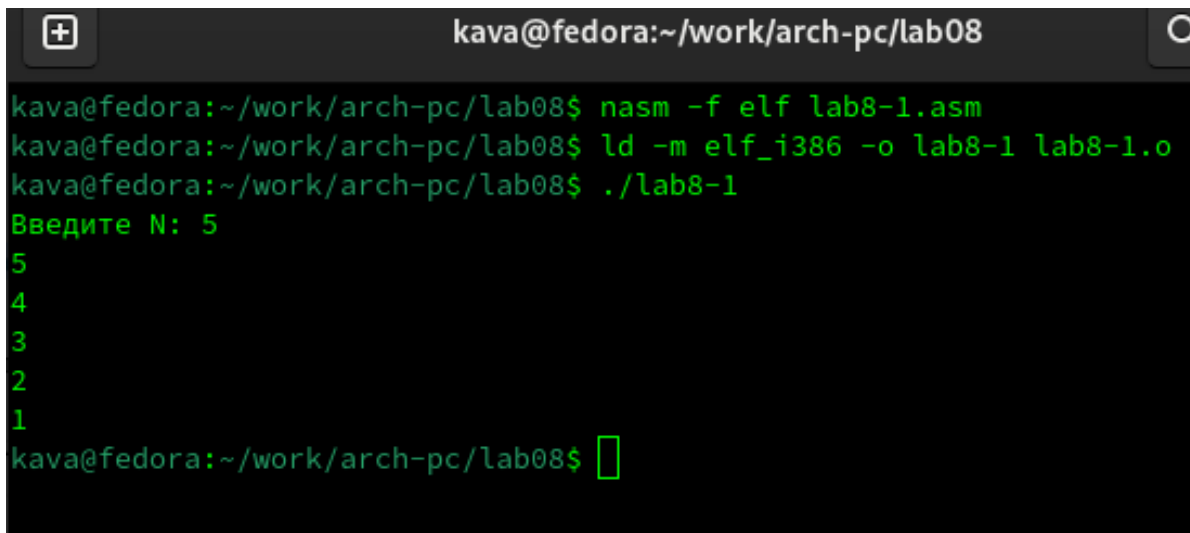
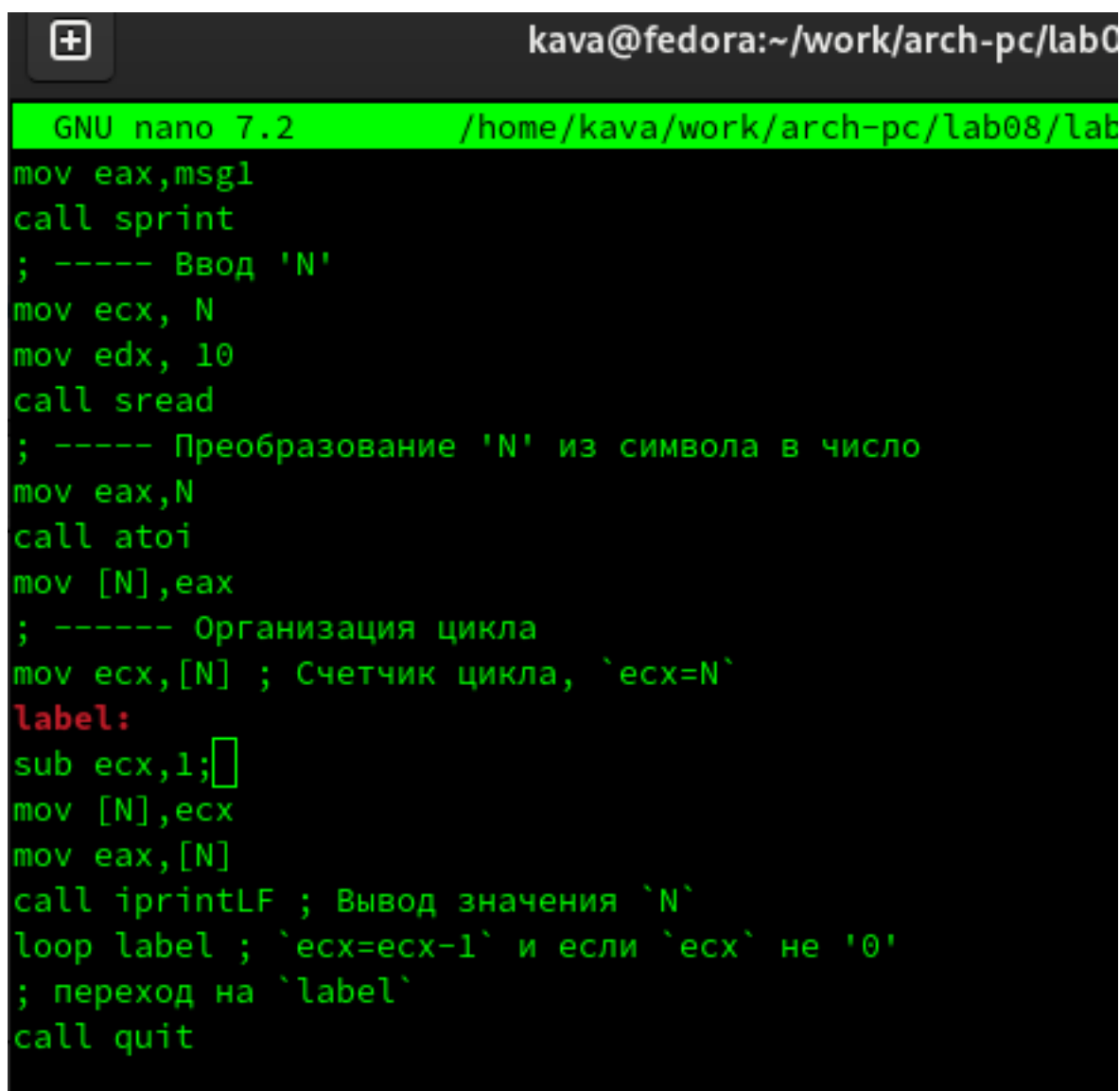


Рис. 2.4: 04

Как видим, она выводит числа от N до единицы включительно. Теперь пробуем изменить код, чтобы в цикле также отнималась единица.

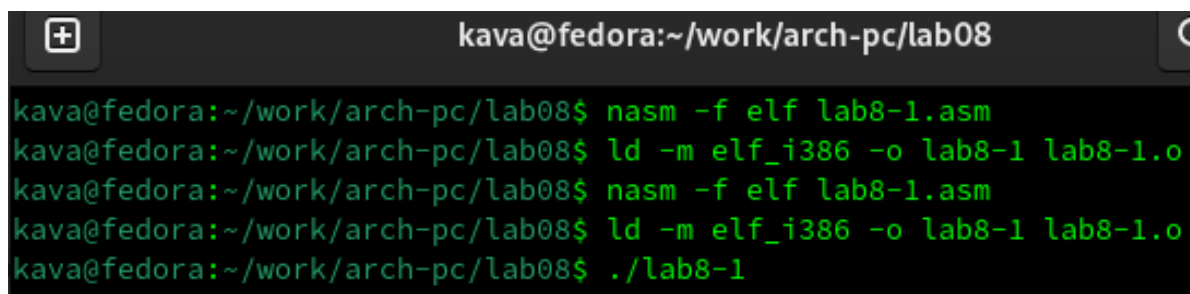


A terminal window titled 'kava@fedora:~/work/arch-pc/lab08' showing the GNU nano 7.2 editor. The editor displays assembly code for a program that prints a message, reads a number 'N', and prints it in a loop. The code is as follows:

```
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;
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. 2.5: 05

Собираем файл и запускаем ее.

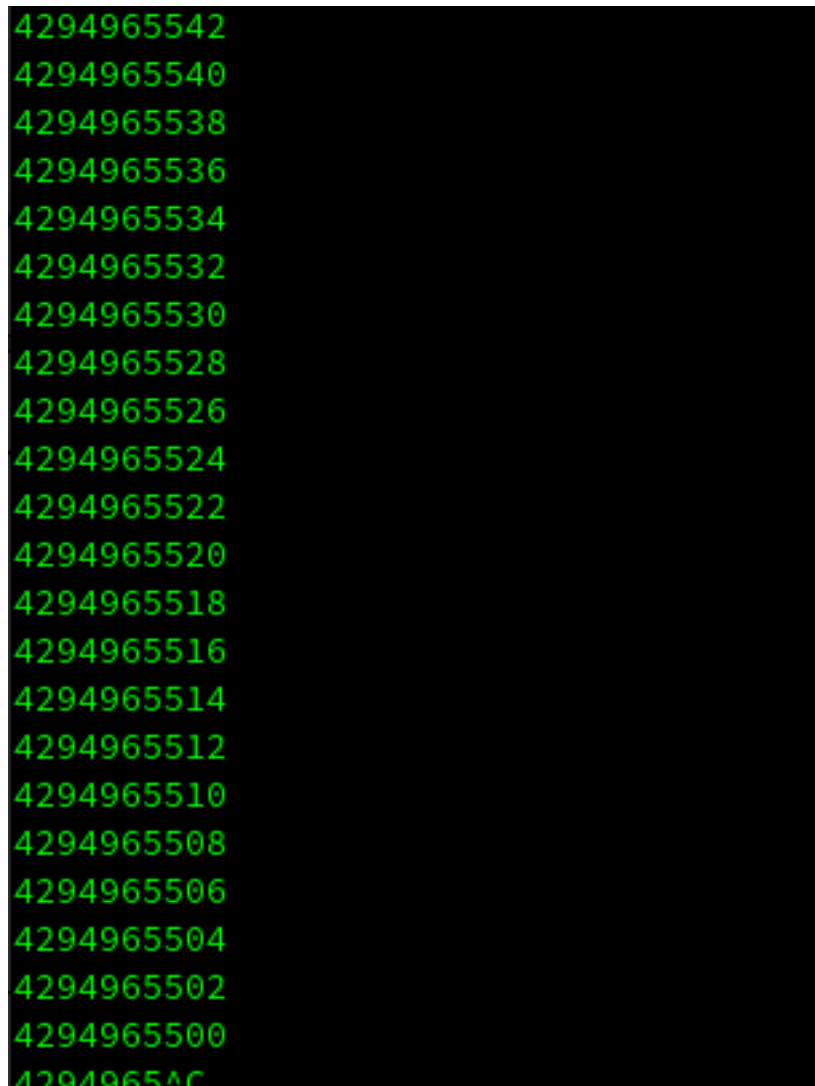


A terminal window titled 'kava@fedora:~/work/arch-pc/lab08' showing the compilation and execution of the assembly file 'lab8-1.asm'. The commands and their outputs are as follows:

```
kava@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
kava@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
kava@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
kava@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
kava@fedora:~/work/arch-pc/lab08$ ./lab8-1
```

Рис. 2.6: 06

Введём в качестве N число 5 и посмотрим на результат выполнения.



```
4294965542
4294965540
4294965538
4294965536
4294965534
4294965532
4294965530
4294965528
4294965526
4294965524
4294965522
4294965520
4294965518
4294965516
4294965514
4294965512
4294965510
4294965508
4294965506
4294965504
4294965502
4294965500
4294965498
```

Рис. 2.7: 07

Цикл выполняется бесконечно, если входное число нечетное, потому что условие остановки цикла $esx=0$ никогда не будет достигнуто. Это происходит из-за того, что регистр esx уменьшается на 2 за каждую интеграцию. Если же входное число четное, то цикл выполняется $N/2$ раз, выводя числа в порядке убавления от $N-1$ до 1 с шагом 2.

```
kava@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 12
11
9
7
5
3
1
kava@fedora:~/work/arch-pc/lab08$
```

Рис. 2.8: 08

Теперь пробуем изменить программу так, чтобы она сохраняла значение регистра `esx` в стек.

```

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 значение ecx в стек
sub 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'
; переход на `label`
call quit
□

```

Рис. 2.9: 09

Пробуем собрать и запустить программу.

```

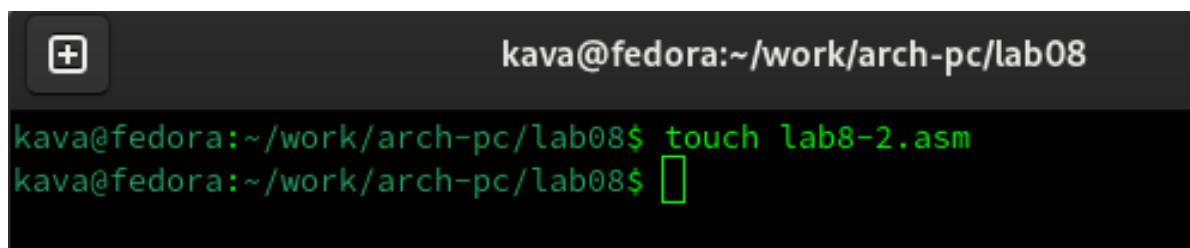
kava@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
kava@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
kava@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 5
4
3
2
1
0
kava@fedora:~/work/arch-pc/lab08$ □

```

Рис. 2.10: 10

Теперь программа выводит все числа от N-1 до нуля, далее создаем второй

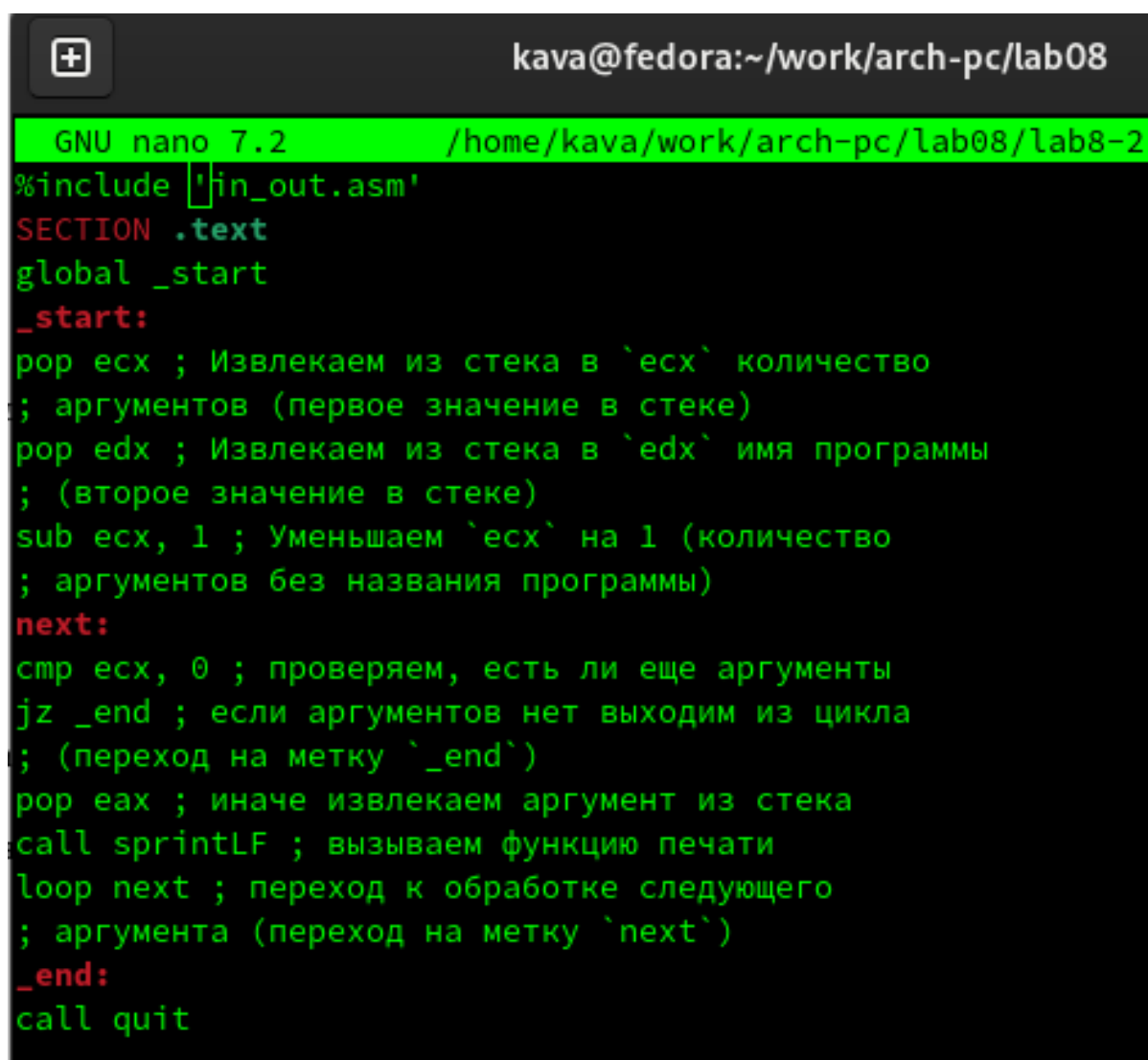
файл.



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

Рис. 2.11: 11

Затем вставляем код из файла листинга 8.2



```
GNU nano 7.2 /home/kava/work/arch-pc/lab08/lab8-2
%include 'in_out.asm'
SECTION .text
global _start
_start:
пор есх ; Извлекаем из стека в `есх` количество
; аргументов (первое значение в стеке)
пор edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub есх, 1 ; Уменьшаем `есх` на 1 (количество
; аргументов без названия программы)
next:
сmp есх, 0 ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
пор еах ; 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. 2.12: 12

Соберем и запустим его указав некоторые аргументы.

```
kava@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
kava@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
kava@fedora:~/work/arch-pc/lab08$ ./lab8-2
kava@fedora:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент2 аргумент3
аргумент1
аргумент2
аргумент3
kava@fedora:~/work/arch-pc/lab08$
```

Рис. 2.13: 13

Создадим третий файл.

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

Рис. 2.14: 14

И вставляем в него код из листинга 8.3. Данная программа находит сумму всех аргументов.

```

GNU nano 7.2 /home/kava/work/arch-pc/lab08/lab8-3.asm
%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

```

Рис. 2.15: 15

Теперь собираем файл и запускаем его.

```

kava@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
kava@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
kava@fedora:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
kava@fedora:~/work/arch-pc/lab08$ 

```

Рис. 2.16: 16

Как видим программа выводит сумму всех аргументов. Изменим её так, чтобы она находила не сумму, а произведение всех аргументов

```

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
mul esi
mov esi,eax
loop next ; переход к обработке следующего аргумента
_end:
mov ebx, eax
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, ebx ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы

```

Рис. 2.17: 17

Собираем программу и запускаем ее.

```

kava@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
kava@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
kava@fedora:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 54600
kava@fedora:~/work/arch-pc/lab08$ 

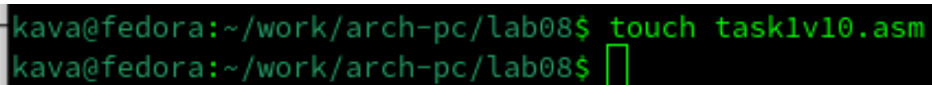
```

Рис. 2.18: 18

Как видим программа выведет правильный ответ.

3 Выполнение задания для самостоятельной работы

Для выполнения самостоятельной работы создадим файл в формате .asm



```
kava@fedora:~/work/arch-pc/lab08$ touch task1v10.asm
kava@fedora:~/work/arch-pc/lab08$
```

Рис. 3.1: 19

В рамках самостоятельной работы необходимо сделать задание под вариантом 10. Там, необходимо сложить результаты выполнения функции $f(x)=5(2+x)$ для всех введённых аргументов.

```

#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=5(2+x)",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, 0       ; Проверяем, есть ли еще аргументы
    jz _end          ; Если аргументов нет, выходим из цикла (переход на метку `_end`)
    pop eax          ; Извлекаем следующий аргумент из стека
    call atoi        ; Преобразуем символ в число
    ; Применяем новую формулу f(x) = 5(2 + x)
    add eax, 2       ; Добавляем 2 к аргументу
    mov ebx, 5       ; Умножаем на 5
    mul ebx
    add esi, eax     ; Добавляем к промежуточной сумме результат: `esi = esi + eax`
    loop next        ; Переход к обработке следующего аргумента
_end:
    mov eax, msg2     ; Вывод сообщения "Функция: f(x)=5(2+x)"
    call sprintfLF
    mov eax, msg      ; Вывод сообщения "Результат: "
    call sprintf
    mov eax, esi      ; Записываем сумму в регистр `eax`
    call iprintfLF    ; Печать результата
    call quit         ; Завершение программы

```

Рис. 3.2: 20

Собираем и запускаем программу, вводя различные аргументы.

```

kava@fedora:~/work/arch-pc/lab08$ nasm -f elf task1v10.asm
kava@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o task1v10 task1v10.o
kava@fedora:~/work/arch-pc/lab08$ ./task1v10 3 4 3 5
Функция: f(x)=5(2+x)
Результат: 115
kava@fedora:~/work/arch-pc/lab08$ ./task1v10 1 6 7 11
Функция: f(x)=5(2+x)
Результат: 165
kava@fedora:~/work/arch-pc/lab08$ 

```

Рис. 3.3: 21

Пересчитав результат вручную, убеждаемся что программа работает верно.

4 Выводы

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