

Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки.

Селиванов Вячеслав Алексеевич

Содержание

| | | |
|----------|-------------------------------------------------|-----------|
| 1 | Цель работы | 4 |
| 2 | Задание | 5 |
| 3 | Теоретическое введение | 6 |
| 4 | Выполнение лабораторной работы | 7 |
| 4.1 | Реализация циклов в NASM | 7 |
| 4.2 | Обработка аргументов командной строки | 13 |
| 4.3 | Задание для самостоятельной работы | 18 |
| 5 | Выводы | 21 |
| | Список литературы | 22 |

Список иллюстраций

| | | |
|------|----------------------------------------|----|
| 4.1 | Создание каталога | 7 |
| 4.2 | Перемещение по директории | 7 |
| 4.3 | Создание файла | 7 |
| 4.4 | Копирование файла in_out.asm | 8 |
| 4.5 | Редактирование программы | 9 |
| 4.6 | Запуск программы | 10 |
| 4.7 | Редактирование программы | 11 |
| 4.8 | Запуск программы | 11 |
| 4.9 | Редактирование программы | 12 |
| 4.10 | Запуск программы | 13 |
| 4.11 | Создание файла | 13 |
| 4.12 | Редактирование программы | 14 |
| 4.13 | Запуск программы | 14 |
| 4.14 | Создание файла | 15 |
| 4.15 | Редактирование программы | 15 |
| 4.16 | Запуск программы | 16 |
| 4.17 | Редактирование программы | 16 |
| 4.18 | Запуск программы | 18 |
| 4.19 | Создание файла | 18 |
| 4.20 | Редактирование программы | 19 |
| 4.21 | Запуск программы | 20 |
| 4.22 | Повторный запуск программы | 20 |

1 Цель работы

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

2 Задание

1. Реализация циклов в NASM
2. Обработка аргументов командной строки
3. Задание 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. 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.

Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается. Для стека существует две основные операции: • добавление элемента в вершину стека (push); • извлечение элемента из вершины стека (pop).

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаю директорию, в которой буду выполнять лабораторную работу (рис. 4.1).

```
[vaselivanov@fedora ~]$ mkdir ~/work/arch-pc/lab08
```

Рис. 4.1: Создание каталога

Перехожу в созданный каталог (рис. 4.2).

```
[vaselivanov@fedora ~]$ cd ~/work/arch-pc/lab08
```

Рис. 4.2: Перемещение по директории

Создаю файл lab8.asm (рис. 4.3). В нём буду делать первое задание.

```
[vaselivanov@fedora lab08]$ touch lab8.asm
```

Рис. 4.3: Создание файла

Также копирую в каталог файл in_out.asm (рис. 4.4). Он понадобится для написания будущих программ.

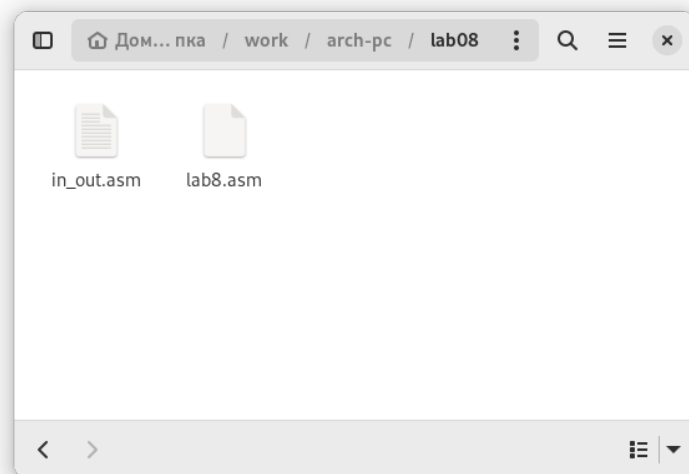
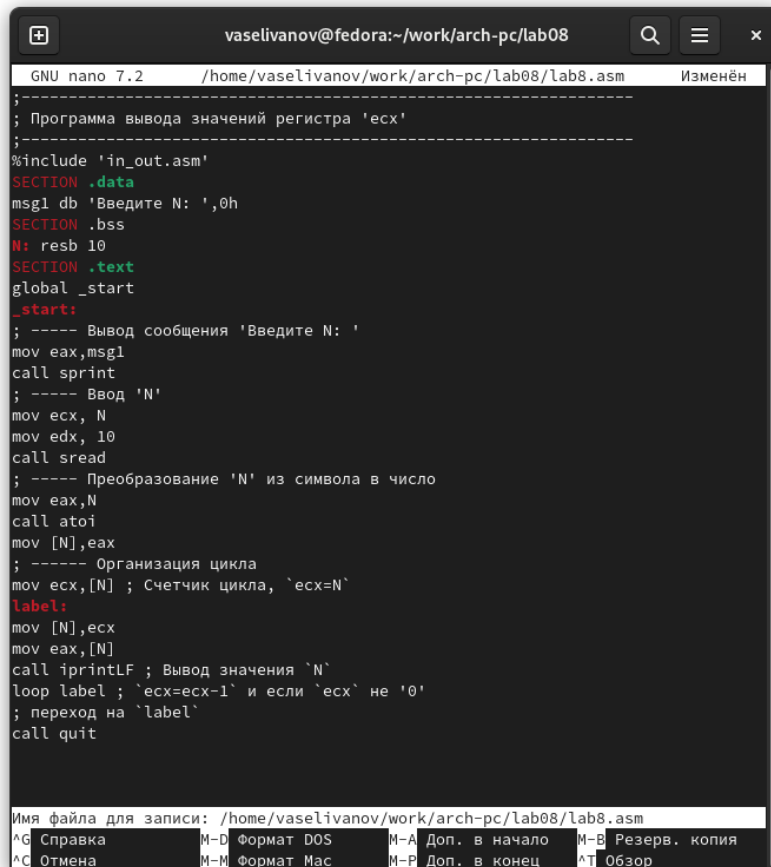


Рис. 4.4: Копирование файла in_out.asm

Записываю текст кода из листинга 8.1 (рис. 4.5). Эта программа запрашивает число N , и выдает все числа перед N вместе с ним до 0.



```
GNU nano 7.2 /home/vaselivanov/work/arch-pc/lab08/lab8.asm
;-----
; 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 ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit

Имя файла для записи: /home/vaselivanov/work/arch-pc/lab08/lab8.asm
^G Справка      M-D Формат DOS   M-A Доп. в начало M-B Резерв. копия
^C Отмена       M-M Формат Mac   M-R Доп. в конец  ^T Обзор
```

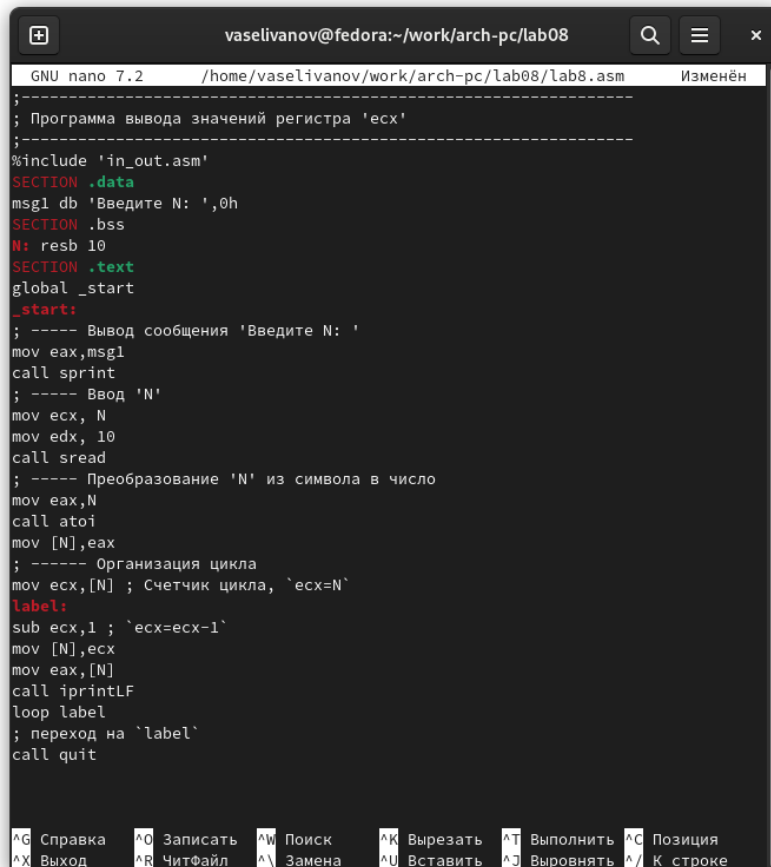
Рис. 4.5: Редактирование программы

Создаю исполняемый код (рис. 4.6).После его запуска убеждаюсь,что программа работает успешно.

```
[vaselivanov@fedora lab08]$ nasm -f elf lab8.asm
[vaselivanov@fedora lab08]$ ld -m elf_i386 -o lab8 lab8.o
[vaselivanov@fedora lab08]$ ./lab8
Введите N: 10
10
9
8
7
6
5
4
3
2
1
[vaselivanov@fedora lab08]$
```

Рис. 4.6: Запуск программы

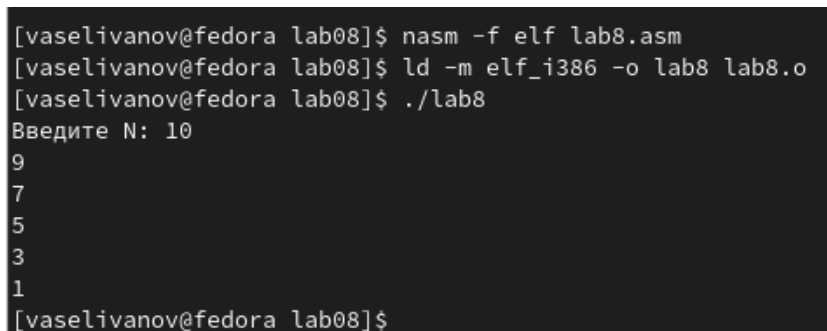
Теперь я редактирую код, добавив изменение значение регистра `ecx` в цикле (рис. 4.7).



```
GNU nano 7.2 /home/vaselivanov/work/arch-pc/lab08/lab8.asm
;
; 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:
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF
loop label
; переход на `label`
call quit
```

Рис. 4.7: Редактирование программы

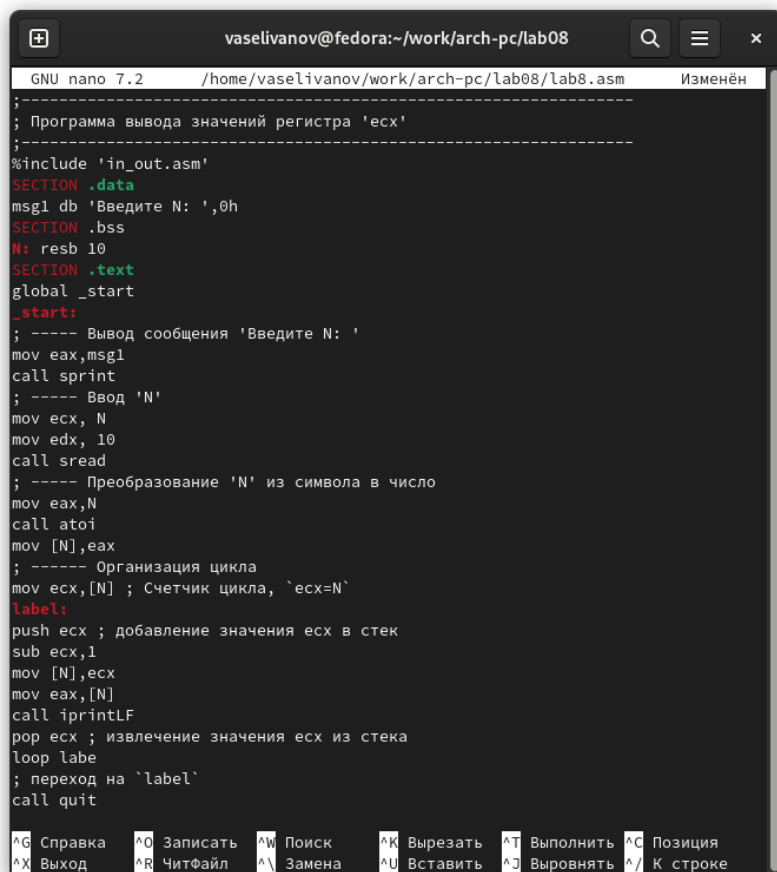
Запускаю программу, она выдаёт предыдущие числа,но перескакивает через 1 (рис. 4.8).



```
[vaselivanov@fedora lab08]$ nasm -f elf lab8.asm
[vaselivanov@fedora lab08]$ ld -m elf_i386 -o lab8 lab8.o
[vaselivanov@fedora lab08]$ ./lab8
Введите N: 10
9
7
5
3
1
[vaselivanov@fedora lab08]$
```

Рис. 4.8: Запуск программы

Еще раз редактирую код программы, добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop (рис. 4.9).



```
GNU nano 7.2 /home/vaselivanov/work/arch-pc/lab08/lab8.asm
;-----
; 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:
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
; переход на `label`
call quit

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выворнять ^_/ К строке
```

Рис. 4.9: Редактирование программы

Создаю и запускаю исполняемый файл (рис. 4.10). Теперь программа показывает все предыдущие числа до 0, не включая заданное N

```
[vaselivanov@fedora lab08]$ nasm -f elf lab8.asm
[vaselivanov@fedora lab08]$ ld -m elf_i386 -o lab8 lab8.o
[vaselivanov@fedora lab08]$ ./lab8
Введите N: 10
9
8
7
6
5
4
3
2
1
0
```

Рис. 4.10: Запуск программы

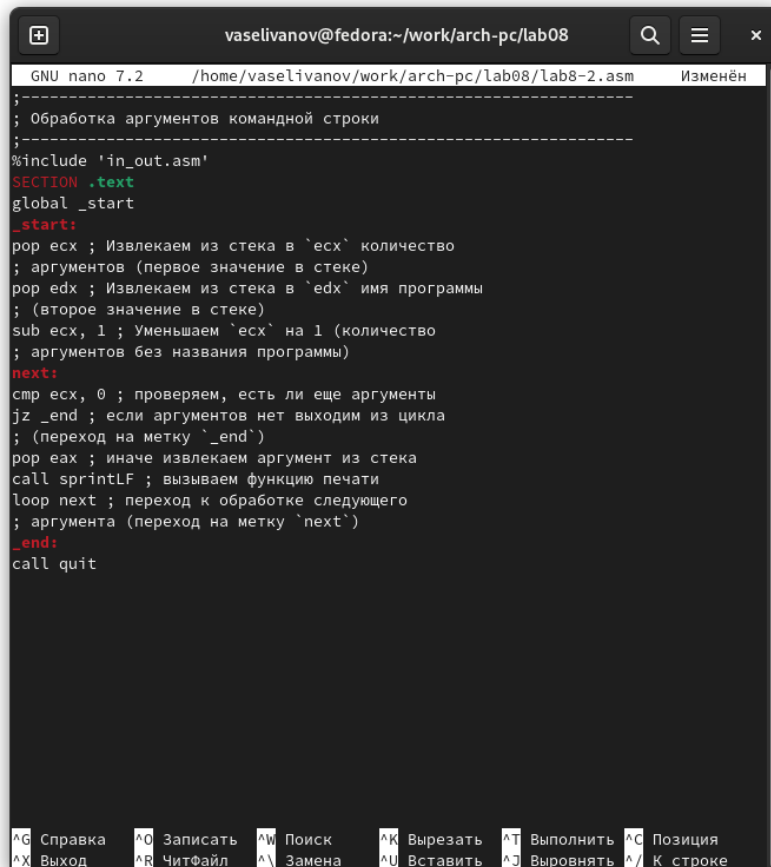
4.2 Обработка аргументов командной строки

Создаю новый файл lab8-2.asm, используя команду touch (рис. 4.11).

```
[vaselivanov@fedora lab08]$ touch lab8-2.asm
```

Рис. 4.11: Создание файла

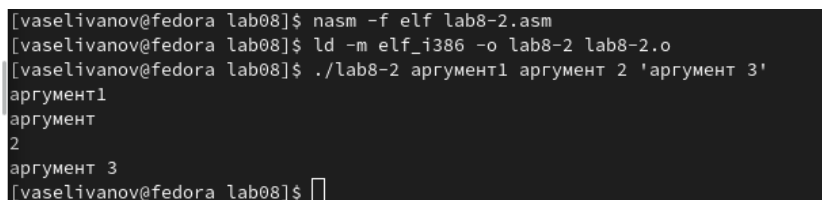
Открываю файл в GNU nano и записываю код из листинга 8.2 (рис. 4.12). Данная программа позволяет выводить на экран аргументы командной строки.



```
GNU nano 7.2 /home/vaselivanov/work/arch-pc/lab08/lab8-2.asm
;-----
; Обработка аргументов командной строки
;-----
#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 printf ; вызываем функцию печати
    loop next ; переход к обработке следующего
             ; аргумента (переход на метку `next`)
_end:
    call _exit
```

Рис. 4.12: Редактирование программы

Запускаю исполняемый файл вместе с аргументами (аргумент1, аргумент2, 'аргумент3') (рис. 4.13).



```
[vaselivanov@fedora lab08]$ nasm -f elf lab8-2.asm
[vaselivanov@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[vaselivanov@fedora lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент 2
аргумент 3
[vaselivanov@fedora lab08]$
```

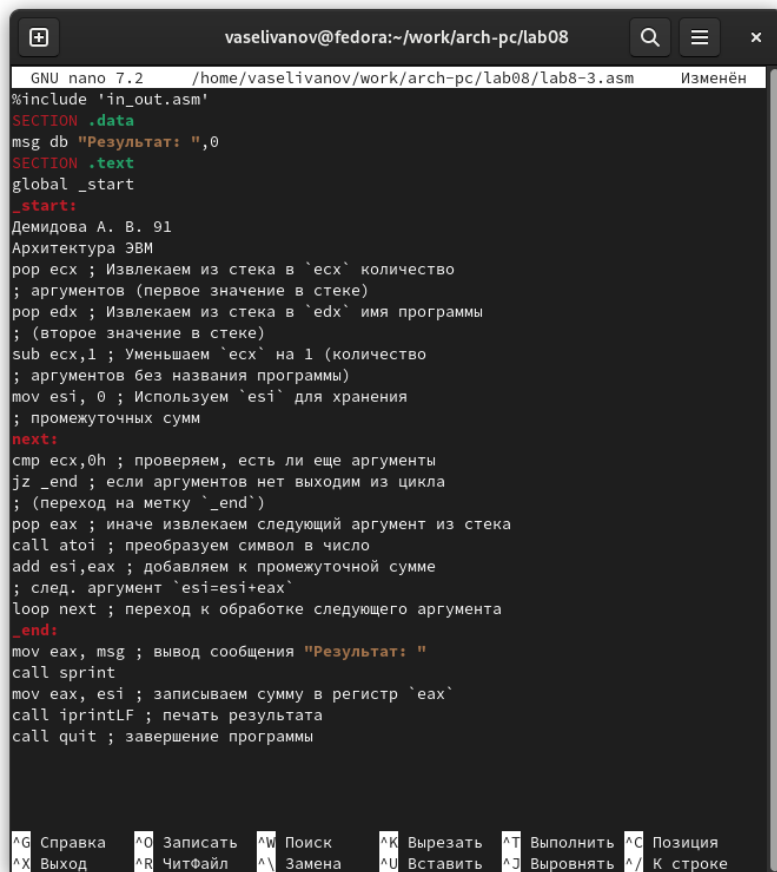
Рис. 4.13: Запуск программы

Создаю новый файл lab8-3.asm, используя команду touch (рис. 4.14).

```
[vaselivanov@fedora lab08]$ touch lab8-3.asm
```

Рис. 4.14: Создание файла

Открываю файл в GNU nano и записываю код из листинга 8.3 (рис. 4.15). Данная программа позволяет выводить на экран сумму аргументов командной строки.



```
GNU nano 7.2 /home/vaselivanov/work/arch-pc/lab08/lab8-3.asm Изменён
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
Демидова А. В. 91
Архитектура ЭВМ
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 iprintf ; печать результата
call quit ; завершение программы

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выворнять ^_/ К строке
```

Рис. 4.15: Редактирование программы

Запускаю исполняемый файл вместе с аргументами (12,13,7,10,5) (рис. 4.16). Программа действительно выдаёт сумму всех аргументов.

```
[vaselivanov@fedora lab08]$ nasm -f elf lab8-3.asm
[vaselivanov@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[vaselivanov@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 47
```

Рис. 4.16: Запуск программы

Теперь редактирую код программы так, чтобы она выводила произведение всех аргументов (рис. 4.17).

```
GNU nano 7.2 /home/vaselivanov/work/arch-pc/lab08/lab8-3.asm Изменён
#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
next:
    cmp ecx,0h ; проверяем, есть ли еще аргументы
    jz _end ; если аргументов нет выходим из цикла
    pop eax ; иначе извлекаем следующий аргумент из стека
    call atoi ; преобразуем символ в числ
    mul esi
    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

^G Справка ^O Записать ^M Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выворнять ^/_ К строке
```

Рис. 4.17: Редактирование программы

```
%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
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в числ
mul esi
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

```

Запускаю исполняемый файл вместе с аргументами (1,3,4,7) (рис. 4.18). Программа выдаёт произведение всех аргументов.

```
[vaselivanov@fedora lab08]$ nasm -f elf lab8-3.asm
[vaselivanov@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[vaselivanov@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 54600
```

Рис. 4.18: Запуск программы

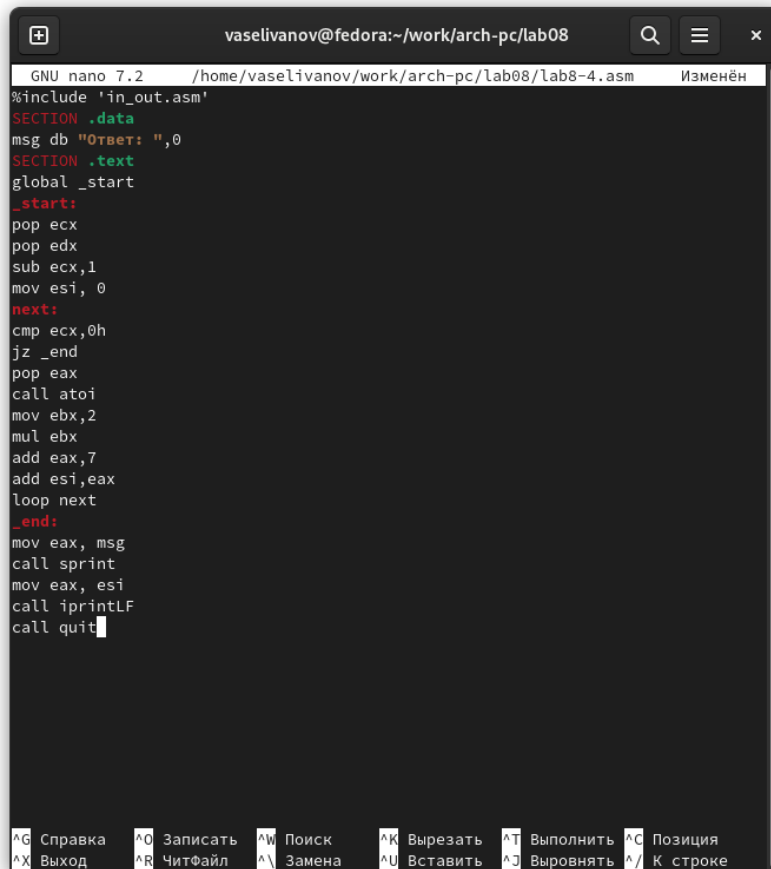
4.3 Задание для самостоятельной работы

Создаю файл lab8-4.asm в котором буду писать код для последней задачи (рис. 4.19).

```
[vaselivanov@fedora lab08]$ touch lab8-4.asm
```

Рис. 4.19: Создание файла

Пишу код программы, который позволяет вывести сумму всех преобразованных аргументов. Преобразования я беру из варианта задания №12 ($2x+7$) (рис. 4.20).



```
GNU nano 7.2 /home/vaselivanov/work/arch-pc/lab08/lab8-4.asm  Изменён
#include 'in_out.asm'
SECTION .data
msg db "Ответ: ",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
mov ebx,2
mul ebx
add eax,7
add esi,eax
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выводить ^_/ К строке

Рис. 4.20: Редактирование программы

```
%include 'in_out.asm'

SECTION .data
msg db "Ответ: ",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
mov ebx,2
mul ebx
add eax,7
add esi,eax
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

```

Запускаю исполняемый файл вместе с аргументами (1,2,3,4) (рис. 4.21). Программа выдаёт верную сумму всех преобразованных аргументов.

```

[vaselivanov@fedora lab08]$ nasm -f elf lab8-4.asm
[vaselivanov@fedora lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[vaselivanov@fedora lab08]$ ./lab8-4 1 2 3 4
Ответ: 48

```

Рис. 4.21: Запуск программы

Повторно запускаю программу, чтобы убедиться, что всё работает верно (рис. 4.22). Программа выдает верный ответ.

```

[vaselivanov@fedora lab08]$ ./lab8-4 9 8 7 6
Ответ: 88

```

Рис. 4.22: Повторный запуск программы

5 Выводы

В данной лабораторной работе я научился работать с циклами, выводом аргументов и функций.

Список литературы

1. Лабораторная работа №8