Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки.

Софич Андрей Геннадьевич

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 15
4.3 Задание для самостоятельной работы 21

5 Выводы **25**

Список литературы **26**

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 12
4.9 Запуск программы 13
4.10 Редактирование программы 14
4.11 Запуск программы 15
4.12 Создание файла 15
4.13 Редактирование программы 16
4.14 Запуск программы 16
4.15 Создание файла 17
4.16 Редактирование программы 17
4.17 Запуск программы 18
4.18 Редактирование программы 19
4.19 Запуск программы 20
4.20 Создание файла 21
4.21 Редактирование программы 22
4.22 Запуск программы 24
4.23 Повторный запуск программы 24

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 указа- тель 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).

Рис. 4.1: Создание каталога

Перехожу в созданный каталог (рис. 4.2).

Рис. 4.2: Перемещение по директории

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

Рис. 4.3: Создание файла

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

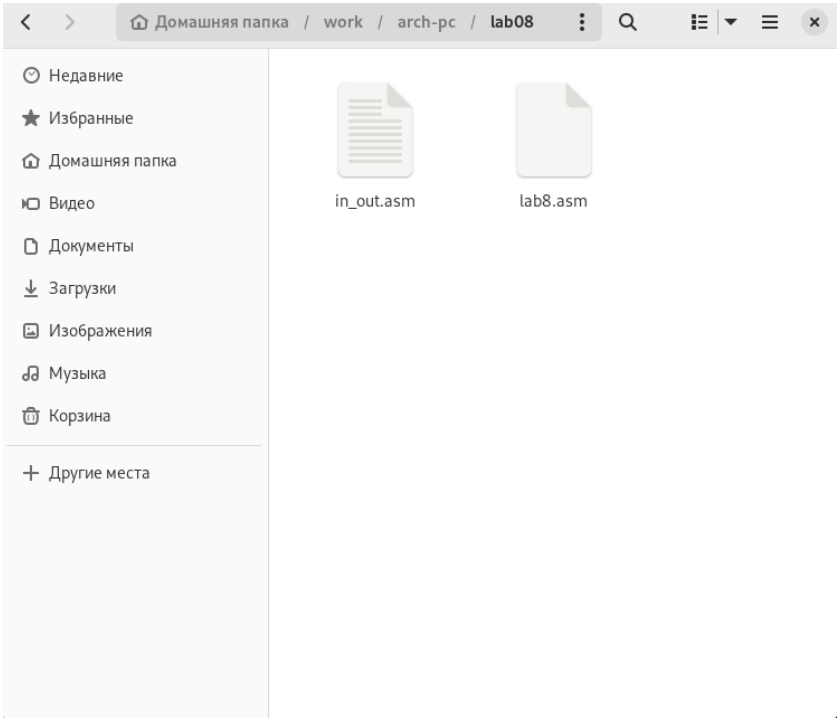

Рис. 4.4: Копирование файла in_out.asm

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.

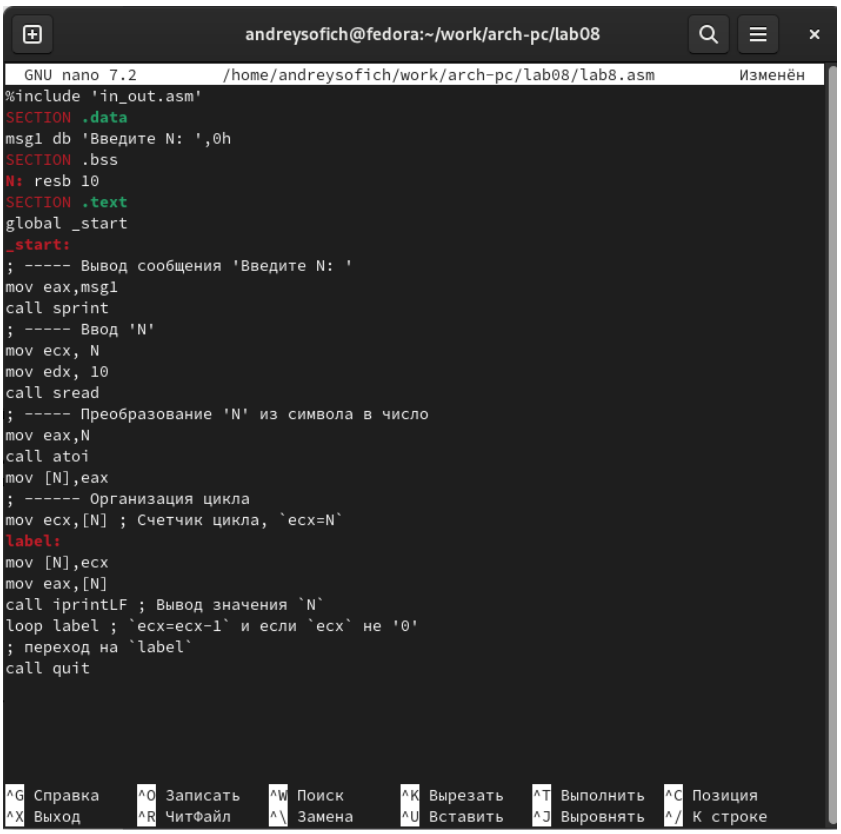

Рис. 4.5: Редактирование программы

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

Рис. 4.6: Запуск программы

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

Рис. 4.7: Редактирование программы

Запускаю программу. Теперь код зацикливается и начинает бесконечно передавать последовательные значения (рис. 4.8).

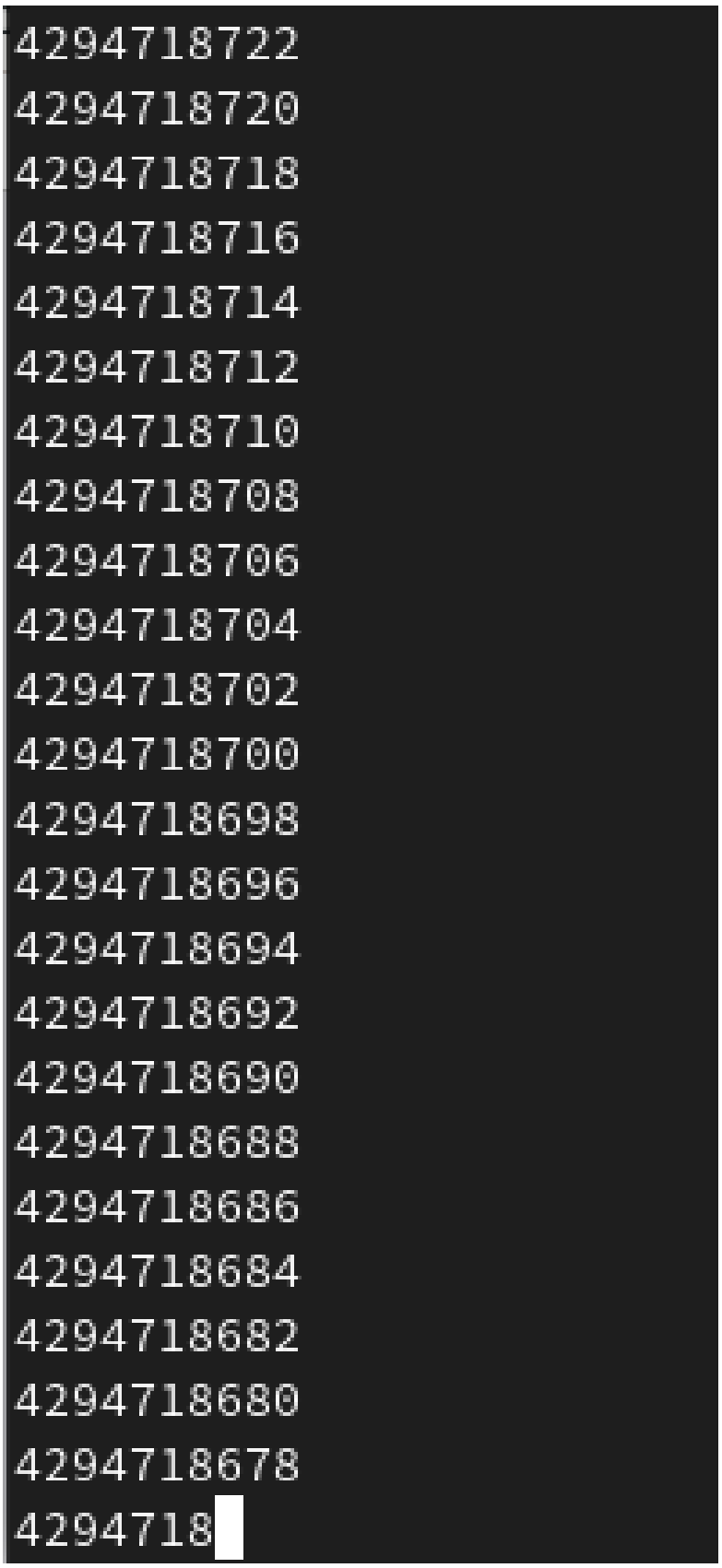

Рис. 4.8: Запуск программы

Пробую еще раз запустить программу,но с другим числом, теперь она выдаёт предыдущие числа,но перескаивает через 1 (рис. 4.9).

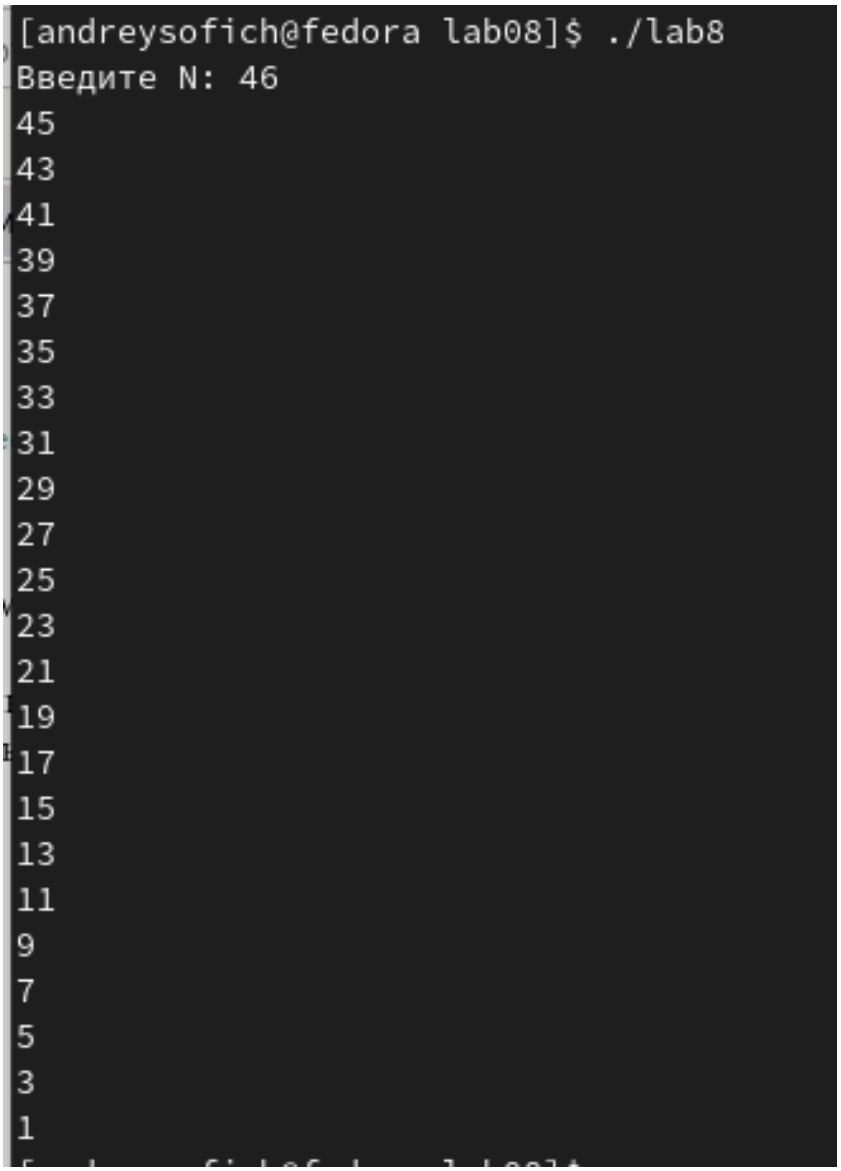

Рис. 4.9: Запуск программы

Еще раз редактирую код программы,добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop (рис. 4.10).

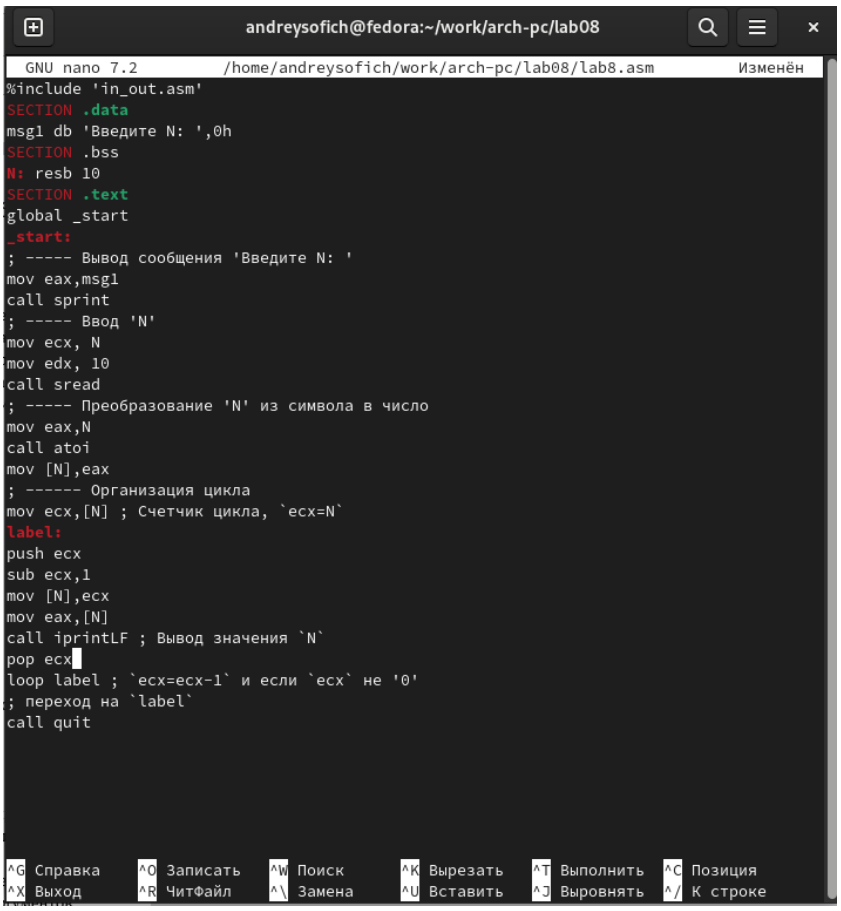

Рис. 4.10: Редактирование программы

Создаю и запускаю исполняемый файл (рис. 4.11). Теперь программа показывает все предыдущие числа до 0,не включая заданное N

```
[andreysofich@fedora lab08]$ nasm -f elf lab8.asm
[andreysofich@fedora lab08]$ ld -m elf_i386 -o lab8  lab8.o
[andreysofich@fedora lab08]$ ./lab8
Введите N: 13
12
11
10
9
8
7
6
5
4
3
2
1
0
```

Рис. 4.11: Запуск программы

4.2 Обработка аргументов командной строки

Создаю новый файл lab8-2.asm, используя команду touch (рис. 4.12).

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

Рис. 4.12: Создание файла

Открываю файл в GNU nano и записываю код из листинга 8.2 (рис. 4.13). Данная программа позволяет выводить на экран аргументы командной строки.

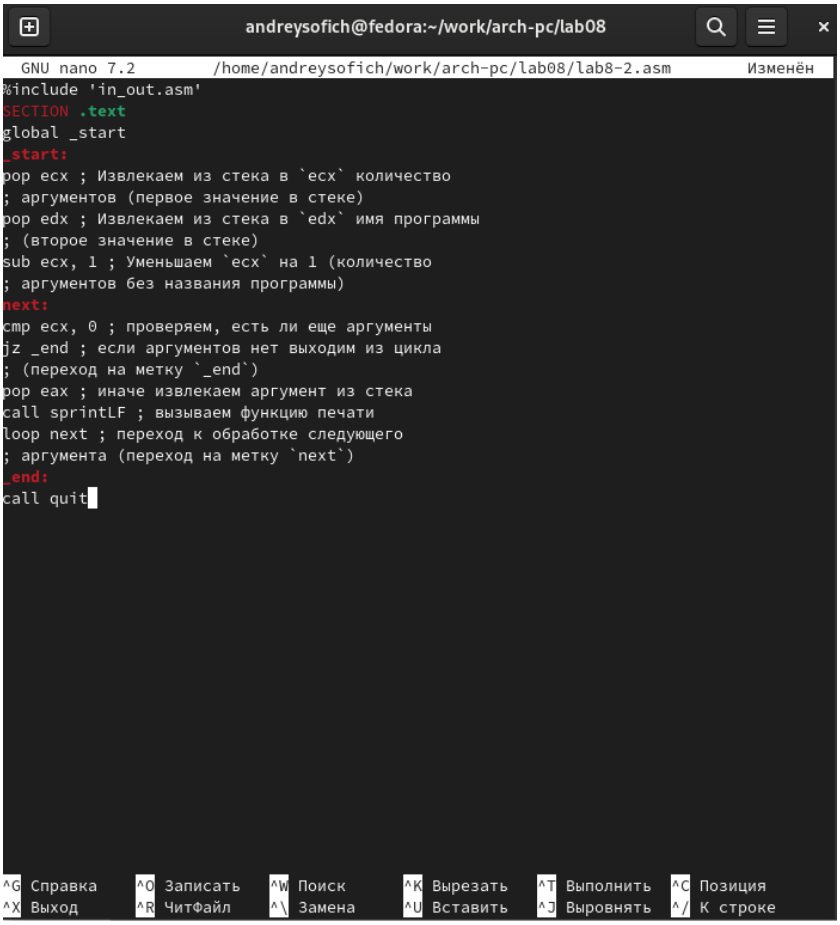

Рис. 4.13: Редактирование программы

Запускаю исполняемый файл вместе с аргументами (аргумент1, аргумент2, ‘аргумент3’) (рис. 4.14).

Рис. 4.14: Запуск программы

Создаю новый файл lab8-3.asm, используя команду touch (рис. 4.15).

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

Рис. 4.15: Создание файла

Открываю файл в GNU nano и записываю код из листинга 8.3 (рис. 4.16). Данная программа позволяет выводить на экран сумму аргументов командной строки.

```
 GNU nano 7.2
%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. 4.16: Редактирование программы

Запускаю исполняемый файл вместе с аргументами (12,13,7,10,5) (рис. 4.17). Программа действительно выдаёт сумму всех аргументов.

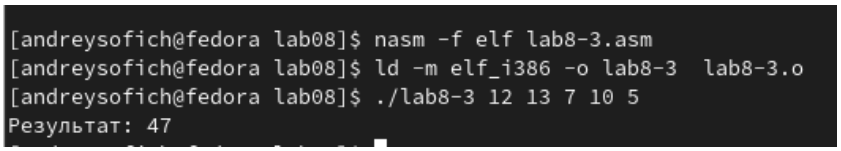

Рис. 4.17: Запуск программы

Теперь редактирую код программы так,чтобы она выводила произведение всех аргументов (рис. 4.18).

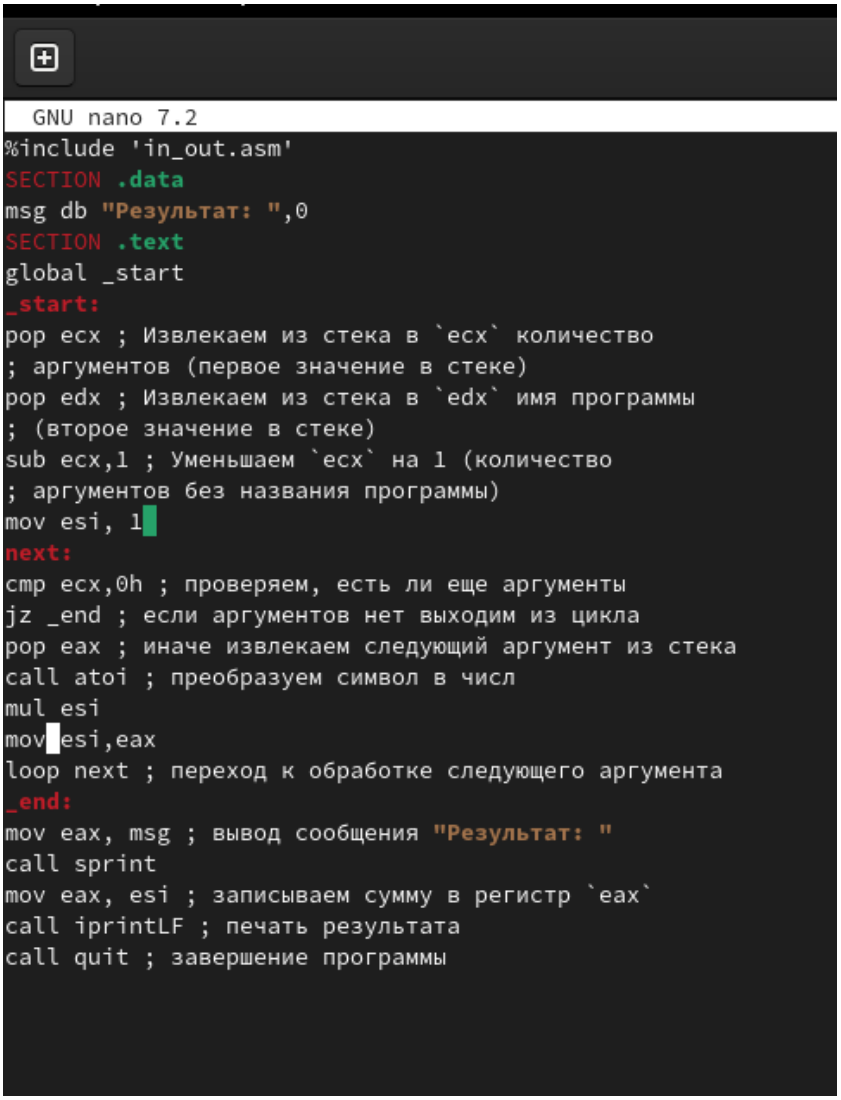

Рис. 4.18: Редактирование программы

```
%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. 4.19). Программа выдаёт произведение всех аргументов.

```
[andreysofich@fedora lab08]$ nasm -f elf lab8-3.asm
[andreysofich@fedora lab08]$ ld -m elf_i386 -o lab8-3  lab8-3.o
[andreysofich@fedora lab08]$ ./lab8-3 1 3 4 7
Результат: 84
```

Рис. 4.19: Запуск программы

4.3 Задание для самостоятельной работы

Создаю файл lab8-4.asm в котором буду писать код для последней задачи (рис. 4.20).

Рис. 4.20: Создание файла

Пишу код программы,который позволяет вывести сумму всех преобразованных аргументов. Преобразования я беру из варианта задания №12 (15x-9) (рис. 4.21).

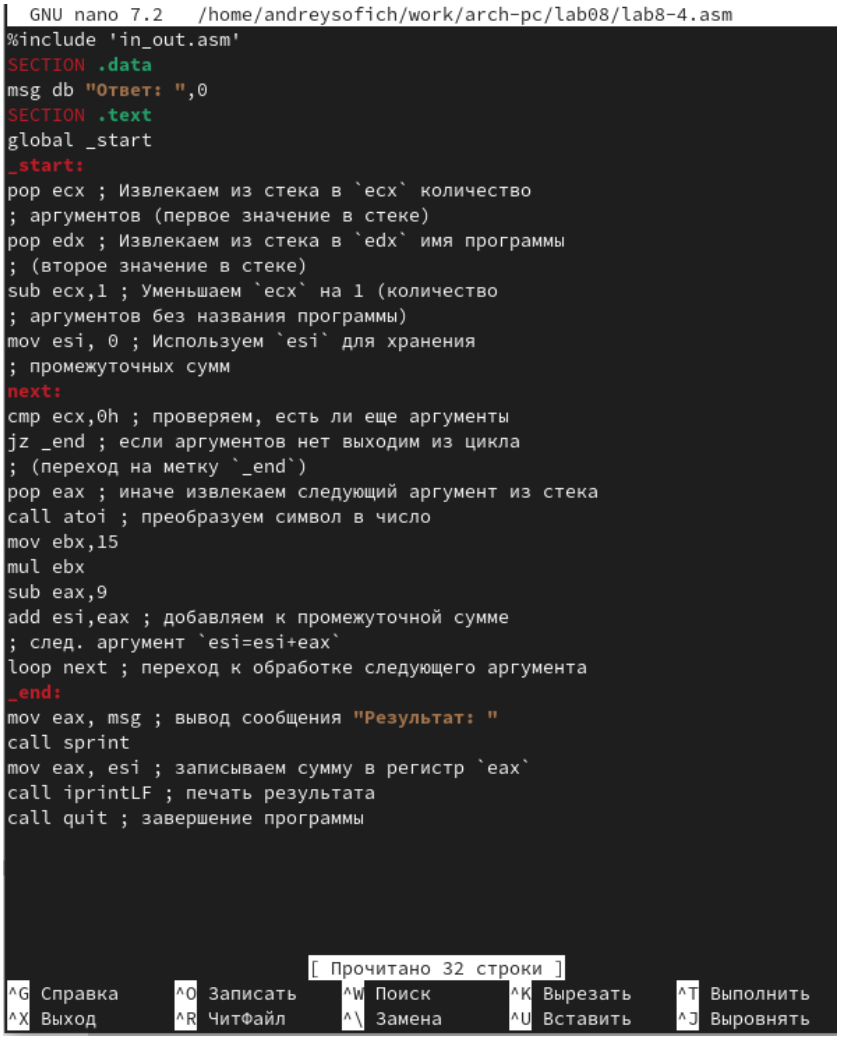

Рис. 4.21: Редактирование программы

```
%include 'in_out.asm'
SECTION .data
msg db "Ответ: ",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
mov ebx,15
mul ebx
sub eax,9
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
```

Запускаю исполняемый файл вместе с аргументами (1,2,3,4) (рис. 4.22). Программа выдаёт верную сумму всех преобразованных аргументов.

```
[andreysofich@fedora lab08]$ nasm -f elf lab8-4.asm
[andreysofich@fedora lab08]$ ld -m elf_i386 -o lab8-4  lab8-4.o
[andreysofich@fedora lab08]$ ./lab8-4 1 2 3 4
Ответ: 114
```

Рис. 4.22: Запуск программы

Повторно запускаю программу,чтобы убедиться, что всё работает верно (рис. 4.23). Программа выдает верный ответ.

```
[andreysofich@fedora lab08]$ ./lab8-4 7 3 5 6
Ответ: 279
```

Рис. 4.23: Повторный запуск программы

5 Выводы

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

Список литературы

1. Лабораторная работа №8