

Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки.

Соколова Александра Олеговна

Содержание

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 аргу-

ментов 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.1).

```
[ssokolova@fedora ~]$ mkdir -p /work/study/2023-2024/Архитектура компьютера/arch-pc/lab08
```

рис.1 Создание каталога

Перехожу в созданный каталог (рис.2).

```
[ssokolova@fedora ~]$ cd ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08  
[ssokolova@fedora lab08]$
```

рис.2 Перемещение по директории

Создаю файл lab8.asm (рис.3). В нём буду делать первое задание.

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

рис.3 Создание файла

Также копирую в каталог файл in_out.asm (рис.4). Он понадобится для написания будущих программ.

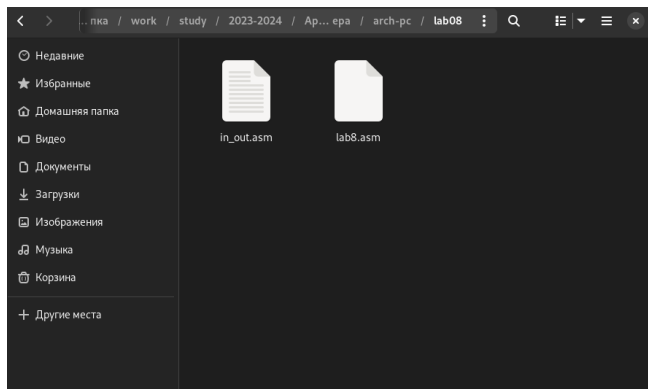


рис.4 Копирование файла in_out.asm

Записываю текст кода из листинга 8.1 (рис.5). Эта программа запрашивает число N, и выдает все числа перед N вместе с ним до 0.

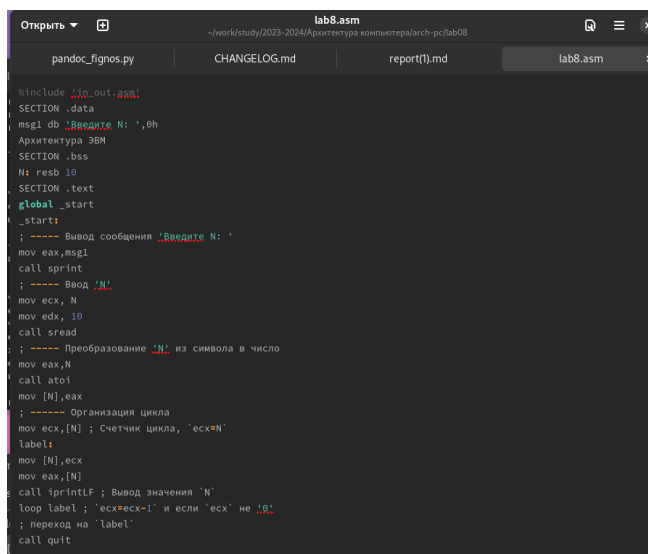


рис.5 Редактирование программы

Создаю исполняемый код (рис.6). После его запуска убеждаюсь, что программа работает успешно.

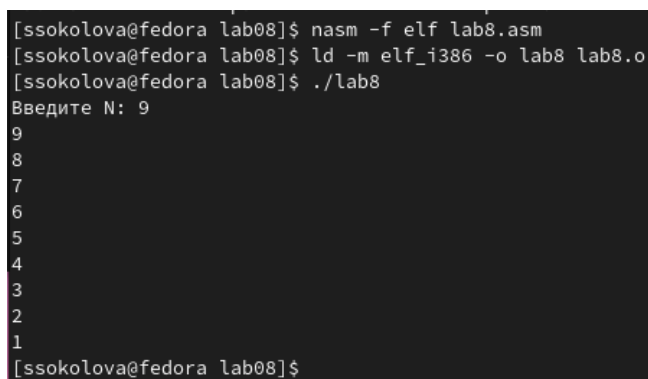
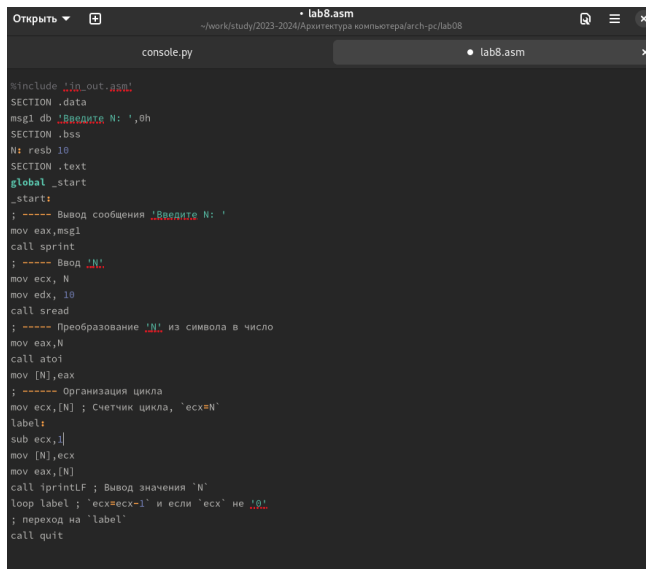


рис.6 Запуск программы

Теперь я редактирую код, добавив изменение значение регистра ecx в цикле (рис.7).



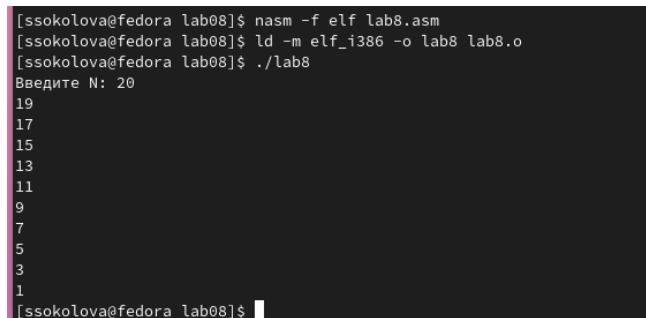
```
Открыть ▾  • lab8.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08

console.py  • lab8.asm x

#include io_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, "еск*N"
label:
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintlnf ; Вывод значения "N"
loop label ; "еск=еск-1" и если "еск" не "0"
; переход на "label"
call quit
```

рис.7 Редактирование программы

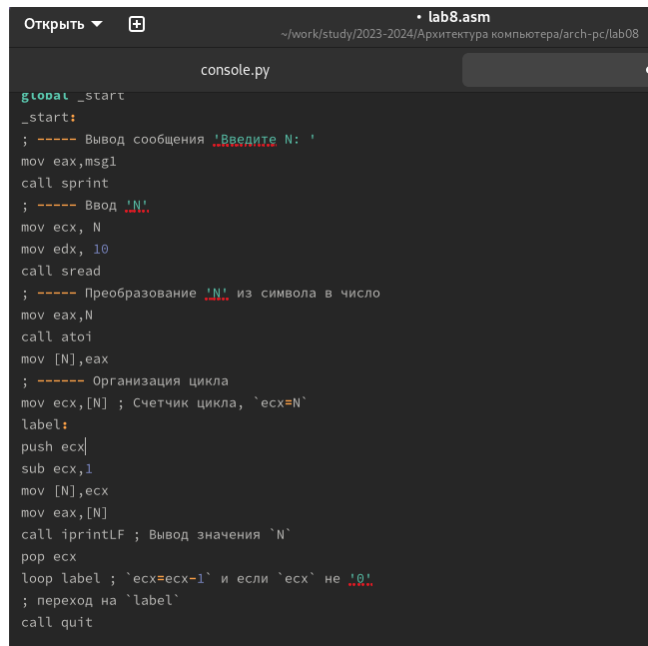
Запускаю программу. Теперь код зацикливается и начинает бесконечно передавать последовательные значения, но перескакивает через 1 (рис.8).



```
[ssokolova@fedora lab08]$ nasm -f elf lab8.asm
[ssokolova@fedora lab08]$ ld -m elf_i386 -o lab8 lab8.o
[ssokolova@fedora lab08]$ ./lab8
Введите N: 20
19
17
15
13
11
9
7
5
3
1
[ssokolova@fedora lab08]$
```

рис.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.10).

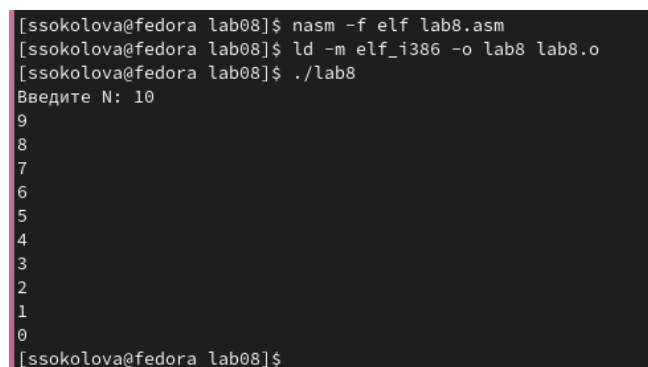


```
Открыть ▾ + lab8.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08
console.py

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
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintlnLF ; Вывод значения 'N'
pop ecx
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit
```

рис.10 Редактирование программы

Создаю и запускаю исполняемый файл (рис.11). Теперь программа показывает все предыдущие числа до 0, не включая заданное N

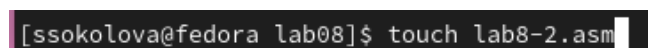


```
[ssokolova@fedora lab08]$ nasm -f elf lab8.asm
[ssokolova@fedora lab08]$ ld -m elf_i386 -o lab8 lab8.o
[ssokolova@fedora lab08]$ ./lab8
Введите N: 10
9
8
7
6
5
4
3
2
1
0
[ssokolova@fedora lab08]$
```

рис.11 Запуск программы

4.2 Обработка аргументов командной строки

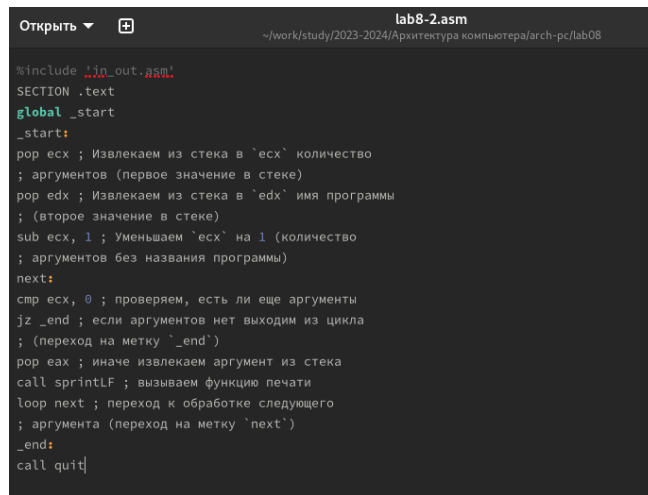
Создаю новый файл lab8-2.asm, используя команду touch (рис.12).



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

рис.12 Создание файла

Открываю файл в текстовом редакторе и записываю код из листинга 8.2 (рис.13). Данная программа позволяет выводить на экран аргументы командной строки.

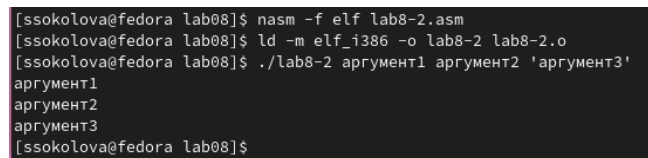


```
lab8-2.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08

#include "io_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 quit
```

рис.13 Редактирование программы

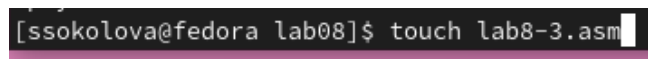
Запускаю исполняемый файл вместе с аргументами (аргумент1, аргумент2, 'аргумент3') (рис.14).



```
[ssokolova@fedora lab08]$ nasm -f elf lab8-2.asm
[ssokolova@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[ssokolova@fedora lab08]$ ./lab8-2 аргумент1 аргумент2 'аргумент3'
аргумент1
аргумент2
аргумент3
[ssokolova@fedora lab08]$
```

рис.14 Запуск программы

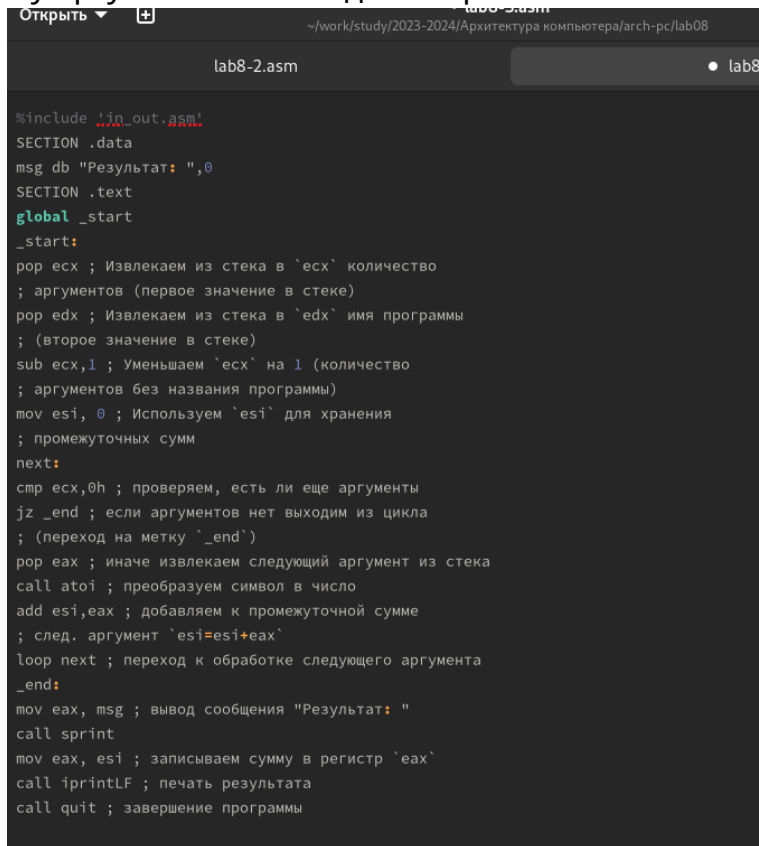
Создаю новый файл lab8-3.asm, используя команду touch (рис.15).



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

рис.15 Создание файла

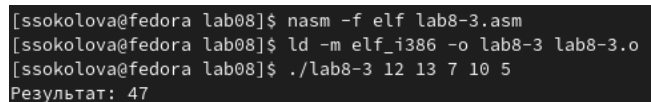
Открываю файл в текстовом редакторе и записываю код из листинга 8.3 (рис.16). Данная программа позволяет выводить на экран сумму аргументов командной строки.



```
lab8-2.asm

%include "io_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
```

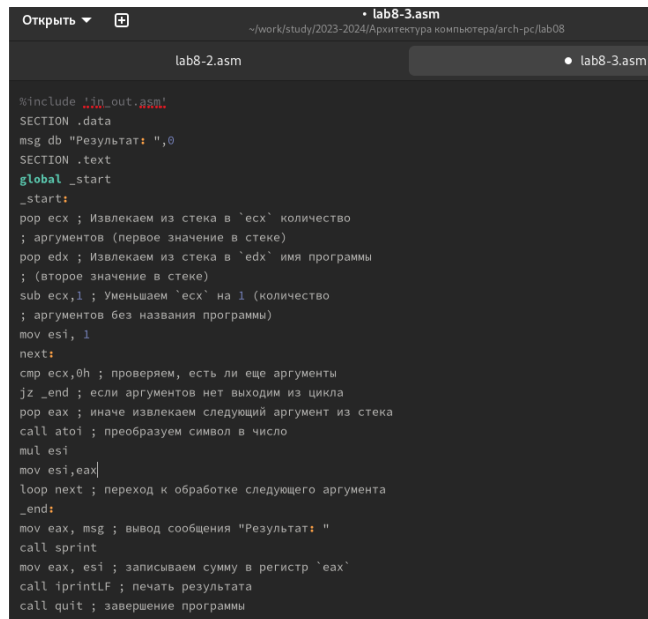
Запускаю исполняемый файл вместе с аргументами (12,13,7,10,5) (рис.17). Программа действительно выдаёт сумму всех аргументов.



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

рис.17 Запуск программы

Теперь редактирую код программы так, чтобы она выводила произведение всех аргументов (рис.18).



```
lab8-2.asm 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, 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 число
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 iprintf ; печать результата
call quit ; завершение программы
```

рис.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.19).
Программа выдаёт произведение всех аргументов.

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

рис.19 Запуск программы

4.3 Задание для самостоятельной работы

Создаю файл lab8-4.asm в котором буду писать код для последней задачи (рис.20).

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

рис.20 Создание файла

Пишу код программы, который позволяет вывести сумму всех преобразованных аргументов. Преобразования я беру из варианта задания №15 $6x+13$ (рис.21).

```
Открыть ▾ + • lab8-4.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08

%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,6
    mul ebx
    add eax,13
    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: "
```

рис.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,6
mul ebx
add eax,13
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.22). Программа выдаёт верную сумму всех преобразованных аргументов.

```

[ssokolova@fedora arch-pc]$ cd lab08/
[ssokolova@fedora lab08]$ nasm -f elf lab8-4.asm
[ssokolova@fedora lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[ssokolova@fedora lab08]$ ./lab8-4 1 2 3 4
Ответ: 112

```

рис.22 Запуск программы

Повторно запускаю программу, чтобы убедиться, что всё работает верно (рис.23). Программа выдает верный ответ.

```
ответ: 112  
[ssokolova@fedora lab08]$ ./lab8-4 6 3 7 1  
ответ: 154  
[ssokolova@fedora lab08]$
```

рис.23 Повторный запуск программы

5 Выводы

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

Список литературы

1. Лабораторная работа №8