

Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютера

Барбакова Алиса Саяновна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
3.1	Организация стека	6
3.2	Инструкции организации циклов	6
4	Выполнение лабораторной работы	8
4.1	Реализация циклов в NASM	8
4.2	Обработка аргументов командной строки	13
4.3	Задание для самостоятельной работы	17
5	Выводы	20
	Список литературы	21

Список иллюстраций

4.1	Создание каталога	8
4.2	Копирование программы из листинга	9
4.3	Запуск программы	10
4.4	Изменение программы	10
4.5	Запуск измененной программы	11
4.6	Добавление push и pop в цикл программы	12
4.7	Запуск измененной программы	12
4.8	Копирование программы из листинга	13
4.9	Запуск второй программы	14
4.10	Копирование программы из третьего листинга	15
4.11	Запуск третьей программы	15
4.12	Изменение третьей программы	16
4.13	Запуск измененной третьей программы	16
4.14	Написание программы для самостоятельной работы	17
4.15	Запуск программы для самостоятельной работы	19

1 Цель работы

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

2 Задание

1. Реализация циклом в NASM
2. Обработка аргументов командной строки
3. Самостоятельное написание программы по материалам лабораторной работы

3 Теоретическое введение

3.1 Организация стека

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 стека существует две основные операции:

- добавление элемента в вершину стека (push);
- извлечение элемента из вершины стека (pop).

3.2 Инструкции организации циклов

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре есх. Наиболее простой является инструкция loop. Она позволяет

организовать безусловный цикл, типичная структура которого имеет следующий вид:

```
mov ecx, 100 ; Количество проходов
```

```
NextStep:
```

```
...
```

```
... ; тело цикла
```

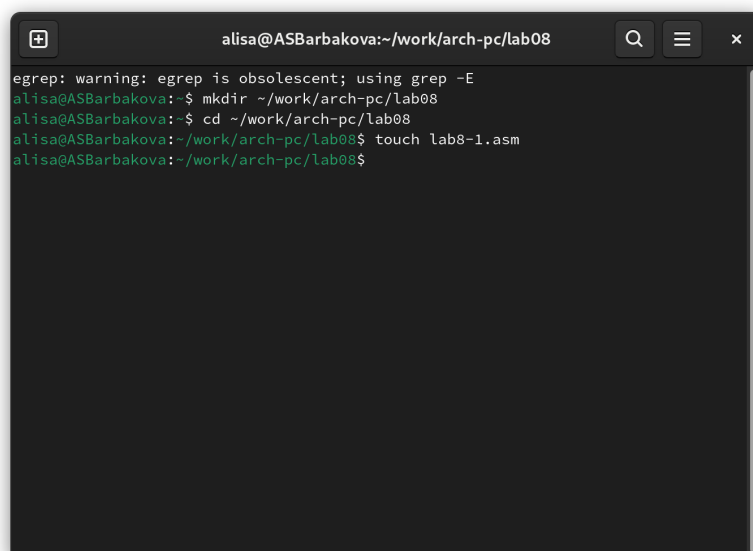
```
...
```

```
loop NextStep ; Повторить `ecx` раз от метки NextStep
```

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

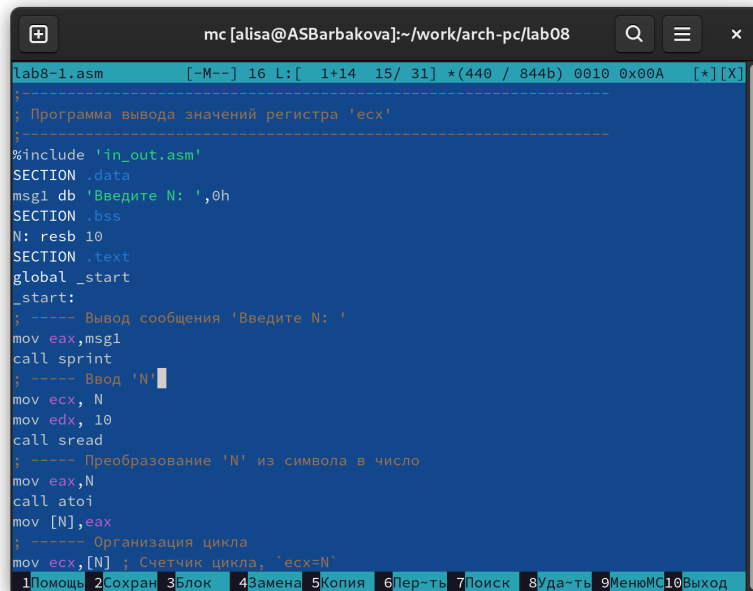
Создаю каталог для программ лабораторной работы №8, перехожу в него и создаю файл lab8-1.asm (рис. -fig. 4.1).



```
alisa@ASBarbakova:~/work/arch-pc/lab08
egrep: warning: egrep is obsolescent; using grep -E
alisa@ASBarbakova:~$ mkdir ~/work/arch-pc/lab08
alisa@ASBarbakova:~$ cd ~/work/arch-pc/lab08
alisa@ASBarbakova:~/work/arch-pc/lab08$ touch lab8-1.asm
alisa@ASBarbakova:~/work/arch-pc/lab08$
```

Рис. 4.1: Создание каталога

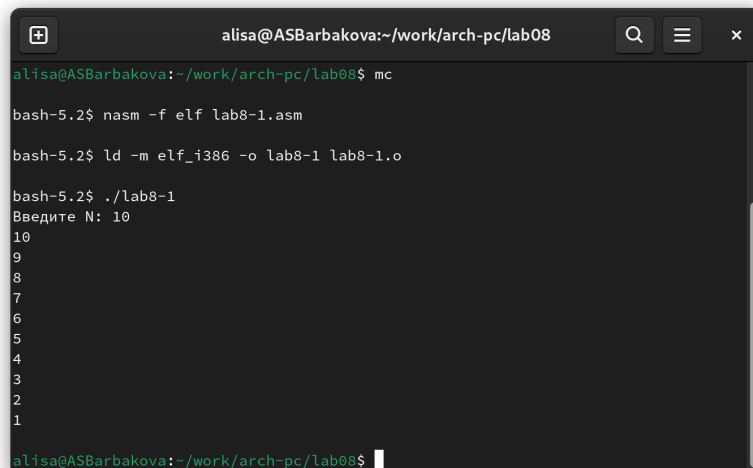
Копирую в созданный файл программу из листинга 8.1 (рис. -fig. 4.2).
Копирую в текущий каталог файл in_out.asm.



```
lab8-1.asm [-M--] 16 L: [ 1+14 15/ 31] *(440 / 844b) 0010 0x00A [*][X]
; 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'
```

Рис. 4.2: Копирование программы из листинга

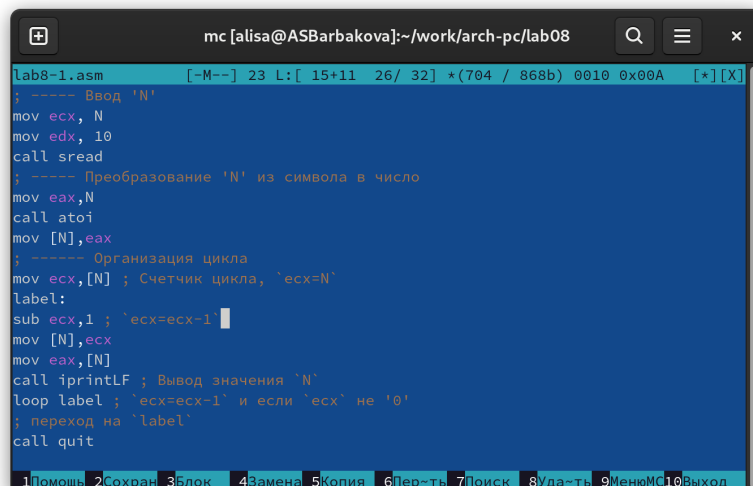
Компилирую и создаю исполняемый файл. Запускаю программу, она показывает работу циклов в NASM (рис. -fig. 4.3).



```
alisa@ASBarbakova:~/work/arch-pc/lab08$ mc
bash-5.2$ nasm -f elf lab8-1.asm
bash-5.2$ ld -m elf_i386 -o lab8-1 lab8-1.o
bash-5.2$ ./lab8-1
Введите N: 10
10
9
8
7
6
5
4
3
2
1
alisa@ASBarbakova:~/work/arch-pc/lab08$
```

Рис. 4.3: Запуск программы

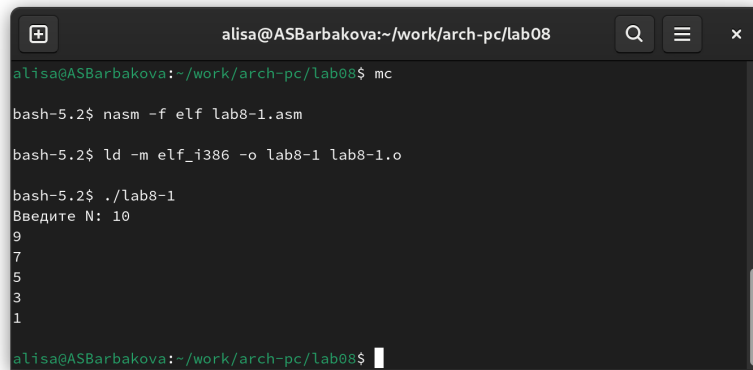
Заменяю изначальный текст программы так, что в теле цикла я изменяю значение регистра `ecx` (рис. -fig. 4.4).



```
lab8-1.asm [-M--] 23 L:[ 15+11 26/ 32] *(704 / 868b) 0010 0x00A [*][X]
; ----- 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 ; Вывод значения 'N'
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit
```

Рис. 4.4: Изменение программы

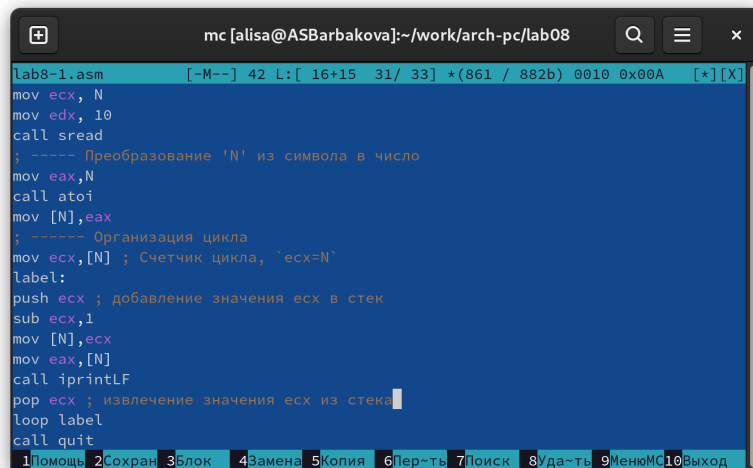
Создаю исполняемый файл, проверяю его работу. Из-за того, что теперь регистр `ecx` на каждой итерации уменьшается на 2 значения, количество итераций уменьшается вдвое (рис. -fig. 4.5).



```
alisa@ASBarbakova:~/work/arch-pc/lab08$ mc
bash-5.2$ nasm -f elf lab8-1.asm
bash-5.2$ ld -m elf_i386 -o lab8-1 lab8-1.o
bash-5.2$ ./lab8-1
Введите N: 10
9
7
5
3
1
alisa@ASBarbakova:~/work/arch-pc/lab08$
```

Рис. 4.5: Запуск измененной программы

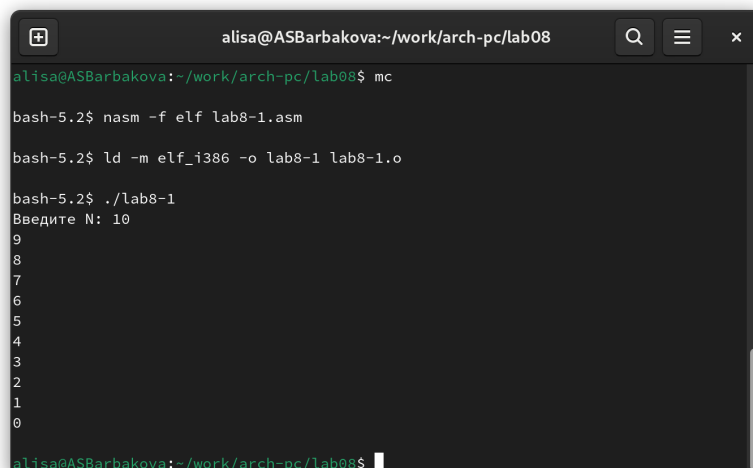
Добавляю команды `push` и `pop` в программу (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop` (рис. -fig. 4.6).



```
lab8-1.asm [-M--] 42 L: [ 16+15 31/ 33] *(861 / 882b) 0010 0x00A [*] [X]
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. 4.6: Добавление push и pop в цикл программы

Запускаю программу (рис. -fig. 4.7). Теперь количество итераций совпадает введенному N, но произошло смещение выводимых чисел на -1.

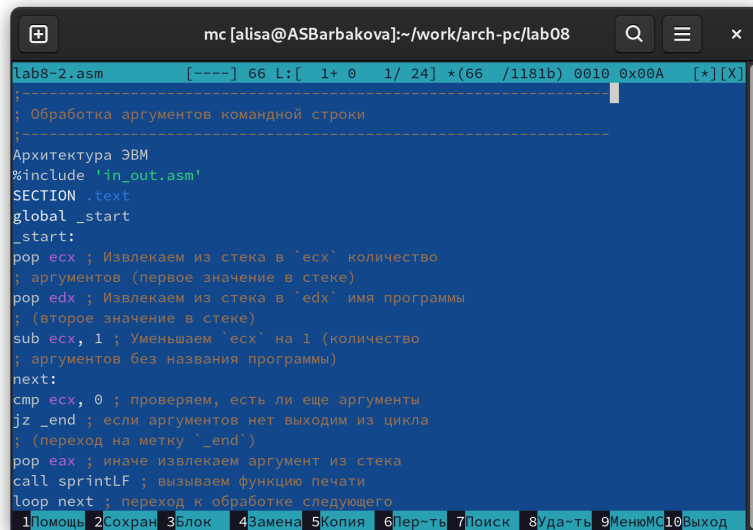


```
alisa@ASBarbakova:~/work/arch-pc/lab08
alisa@ASBarbakova:~/work/arch-pc/lab08$ mc
bash-5.2$ nasm -f elf lab8-1.asm
bash-5.2$ ld -m elf_i386 -o lab8-1 lab8-1.o
bash-5.2$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
alisa@ASBarbakova:~/work/arch-pc/lab08$
```

Рис. 4.7: Запуск измененной программы

4.2 Обработка аргументов командной строки

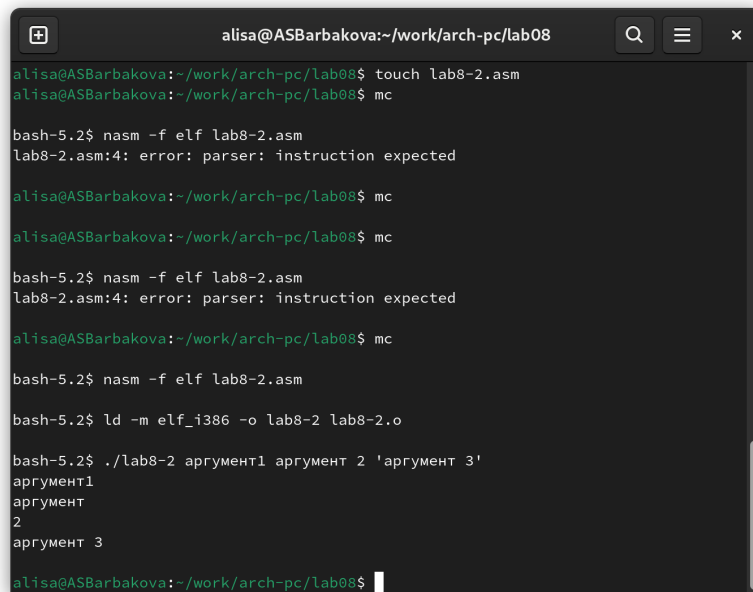
Создаю новый файл lab8-2.asm и копирую в него код из листинга 2 (рис. -fig. 4.8).



```
lab8-2.asm [----] 66 L: [ 1+ 0 1/ 24] *(66 /1181b) 0010 0x00A [*][X]
; Обработка аргументов командной строки
;
Архитектура ЭВМ
#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 sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9МенюMC10Выход
```

Рис. 4.8: Копирование программы из листинга

Компилирую файл и запускаю, указав аргументы. Программой было обработано то же количество аргументов, что и было введено (рис. -fig. 4.9).



```
alisa@ASBarbakova:~/work/arch-pc/lab08
alisa@ASBarbakova:~/work/arch-pc/lab08$ touch lab8-2.asm
alisa@ASBarbakova:~/work/arch-pc/lab08$ mc

bash-5.2$ nasm -f elf lab8-2.asm
lab8-2.asm:4: error: parser: instruction expected

alisa@ASBarbakova:~/work/arch-pc/lab08$ mc

alisa@ASBarbakova:~/work/arch-pc/lab08$ mc

bash-5.2$ nasm -f elf lab8-2.asm
lab8-2.asm:4: error: parser: instruction expected

alisa@ASBarbakova:~/work/arch-pc/lab08$ mc

bash-5.2$ nasm -f elf lab8-2.asm

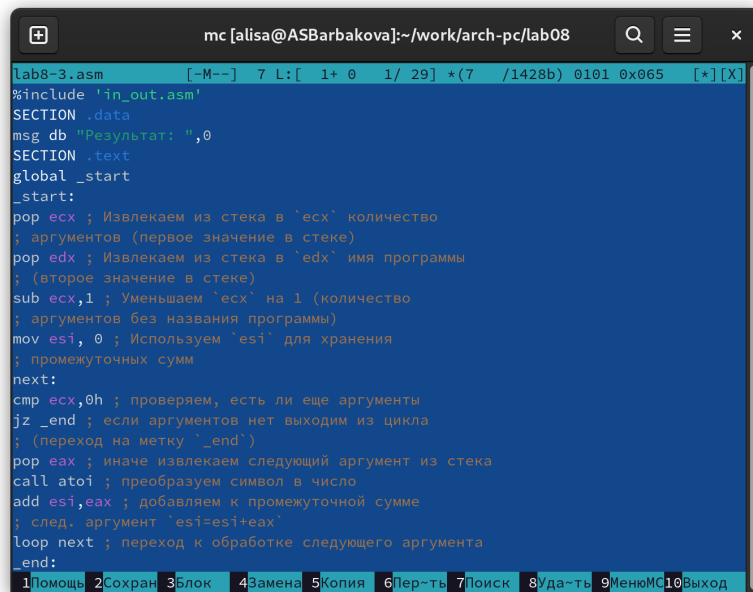
bash-5.2$ ld -m elf_i386 -o lab8-2 lab8-2.o

bash-5.2$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3

alisa@ASBarbakova:~/work/arch-pc/lab08$
```

Рис. 4.9: Запуск второй программы

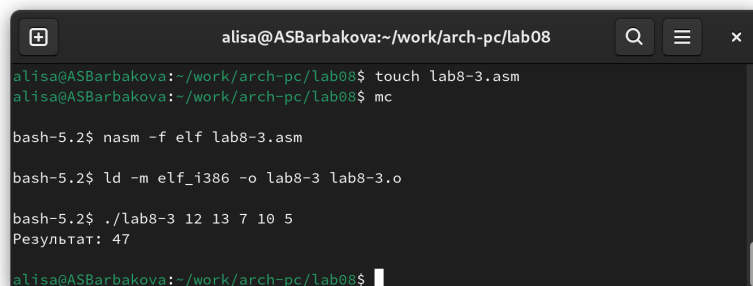
Создаю новый файл lab8-3.asm для программы и копирую в него код из листинга 3 (рис. -fig. 4.10).



```
lab8-3.asm [-M--] 7 L:[ 1+ 0 1/ 29] *(7 /1428b) 0101 0x065 [*][X]
#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:
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9МенюMc10Выход
```

Рис. 4.10: Копирование программы из третьего листинга

Создаю исполняемый файл и запускаю его. Программа выводит сумму чисел, которые передаются в программу как аргументы (рис. -fig. 4.11).



```
alisa@ASBarbakova:~/work/arch-pc/lab08$ touch lab8-3.asm
alisa@ASBarbakova:~/work/arch-pc/lab08$ mc

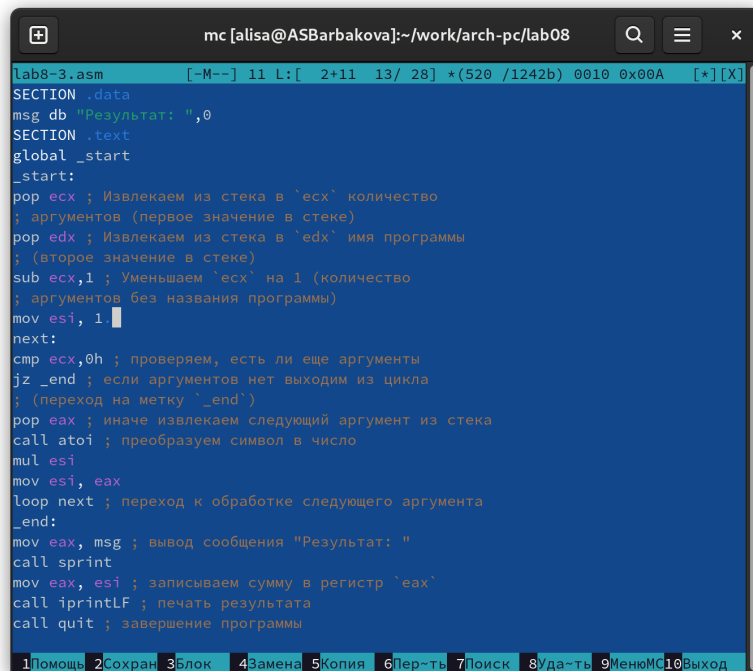
bash-5.2$ nasm -f elf lab8-3.asm

bash-5.2$ ld -m elf_i386 -o lab8-3 lab8-3.o

bash-5.2$ ./lab8-3 12 13 7 10 5
Результат: 47
alisa@ASBarbakova:~/work/arch-pc/lab08$
```

Рис. 4.11: Запуск третьей программы

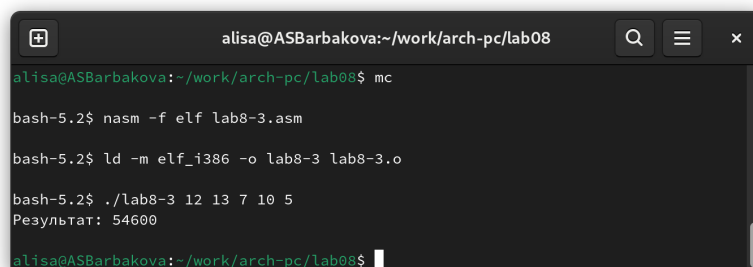
Изменяю текст программы так, чтобы она вычисляла произведение вводимых аргументов (рис. -fig. 4.12).



```
lab8-3.asm [-M--] 11 L: [ 2+11 13/ 28] *(520 /1242b) 0010 0x00A [*] [X]
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
; (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 eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 4.12: Изменение третьей программы

Теперь программа умножает данные на вход числа (рис. -fig. 4.13).

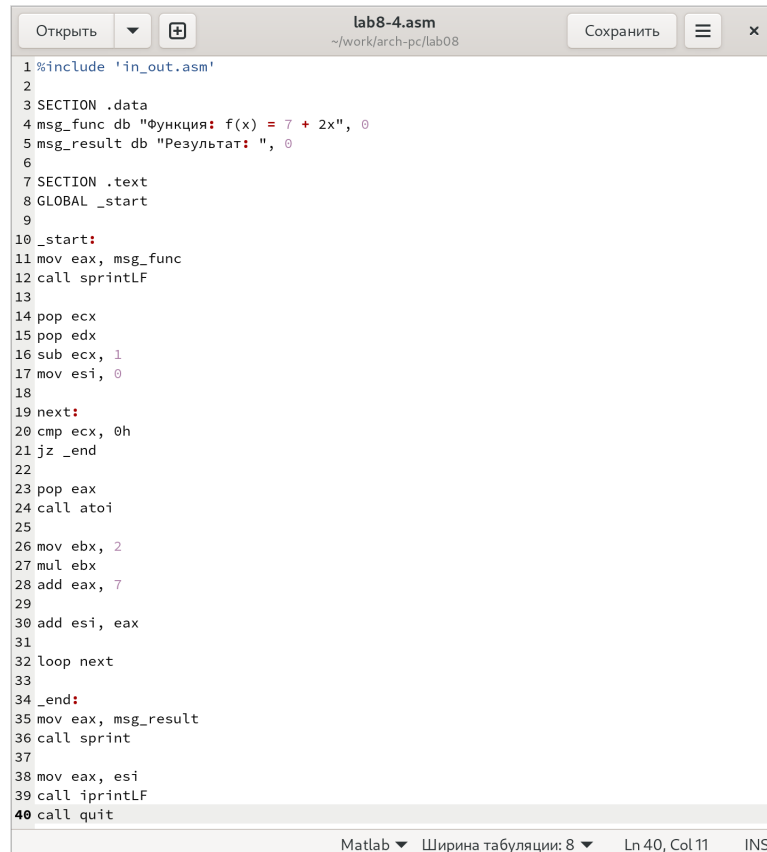


```
alisa@ASBarbakova:~/work/arch-pc/lab08
alisa@ASBarbakova:~/work/arch-pc/lab08$ mc
bash-5.2$ nasm -f elf lab8-3.asm
bash-5.2$ ld -m elf_i386 -o lab8-3 lab8-3.o
bash-5.2$ ./lab8-3 12 13 7 10 5
Результат: 54600
alisa@ASBarbakova:~/work/arch-pc/lab08$
```

Рис. 4.13: Запуск измененной третьей программы

4.3 Задание для самостоятельной работы

Пишу программу, которая будет находить сумму значений для функции $f(x) = 7 + 2x$ из 8-го варианта, полученного в 6-ой лабораторной работе (рис. -fig. 4.14).



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg_func db "Функция: f(x) = 7 + 2x", 0
5 msg_result db "Результат: ", 0
6
7 SECTION .text
8 GLOBAL _start
9
10 _start:
11 mov eax, msg_func
12 call sprintf
13
14 pop ecx
15 pop edx
16 sub ecx, 1
17 mov esi, 0
18
19 next:
20 cmp ecx, 0h
21 jz _end
22
23 pop eax
24 call atoi
25
26 mov ebx, 2
27 mul ebx
28 add eax, 7
29
30 add esi, eax
31
32 loop next
33
34 _end:
35 mov eax, msg_result
36 call sprintf
37
38 mov eax, esi
39 call iprintLF
40 call quit
```

Рис. 4.14: Написание программы для самостоятельной работы

Код программы:

```
%include 'in_out.asm'
.
```

```
SECTION .data.
```

```
msg_func db "Функция:  $f(x) = 7 + 2x$ ", 0.
```

```
msg_result db "Результат: ", 0.
```

```
SECTION .text.
```

```
GLOBAL _start.
```

```
_start:.
```

```
mov eax, msg_func.
```

```
call sprintf
```

```
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_result.
```

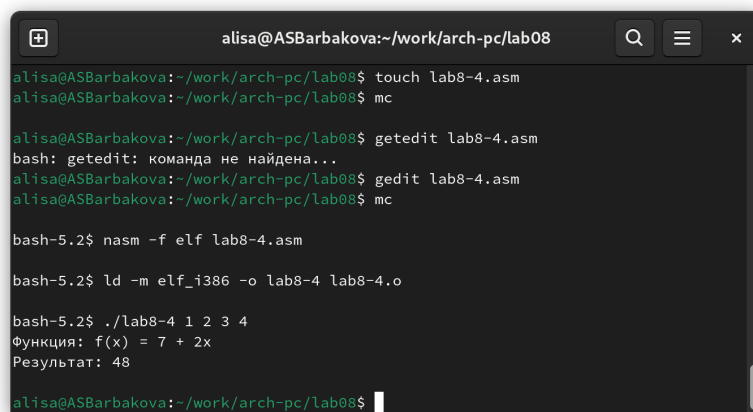
```
call sprint
```

```
mov eax, esi.
```

```
call iprintLF.
```

```
call quit.
```

Проверяю работу программы, указав в качестве аргумента несколько чисел (рис. -fig. 4.15). Программа работает верно.



```
alisa@ASBarbakova:~/work/arch-pc/lab08$ touch lab8-4.asm
alisa@ASBarbakova:~/work/arch-pc/lab08$ mc

alisa@ASBarbakova:~/work/arch-pc/lab08$ getedit lab8-4.asm
bash: getedit: команда не найдена...
alisa@ASBarbakova:~/work/arch-pc/lab08$ gedit lab8-4.asm
alisa@ASBarbakova:~/work/arch-pc/lab08$ mc

bash-5.2$ nasm -f elf lab8-4.asm

bash-5.2$ ld -m elf_i386 -o lab8-4 lab8-4.o

bash-5.2$ ./lab8-4 1 2 3 4
Функция:  $f(x) = 7 + 2x$ 
Результат: 48

alisa@ASBarbakova:~/work/arch-pc/lab08$
```

Рис. 4.15: Запуск программы для самостоятельной работы

5 Выводы

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием циклов а также научилась обрабатывать аргументы командной строки.

Список литературы

1. Курс на ТУИС
2. Лабораторная работа №8