

Отчет по лабораторной работе №8

Дисциплина: архитектура компьютера

Козин Иван Евгеньевич

Содержание

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

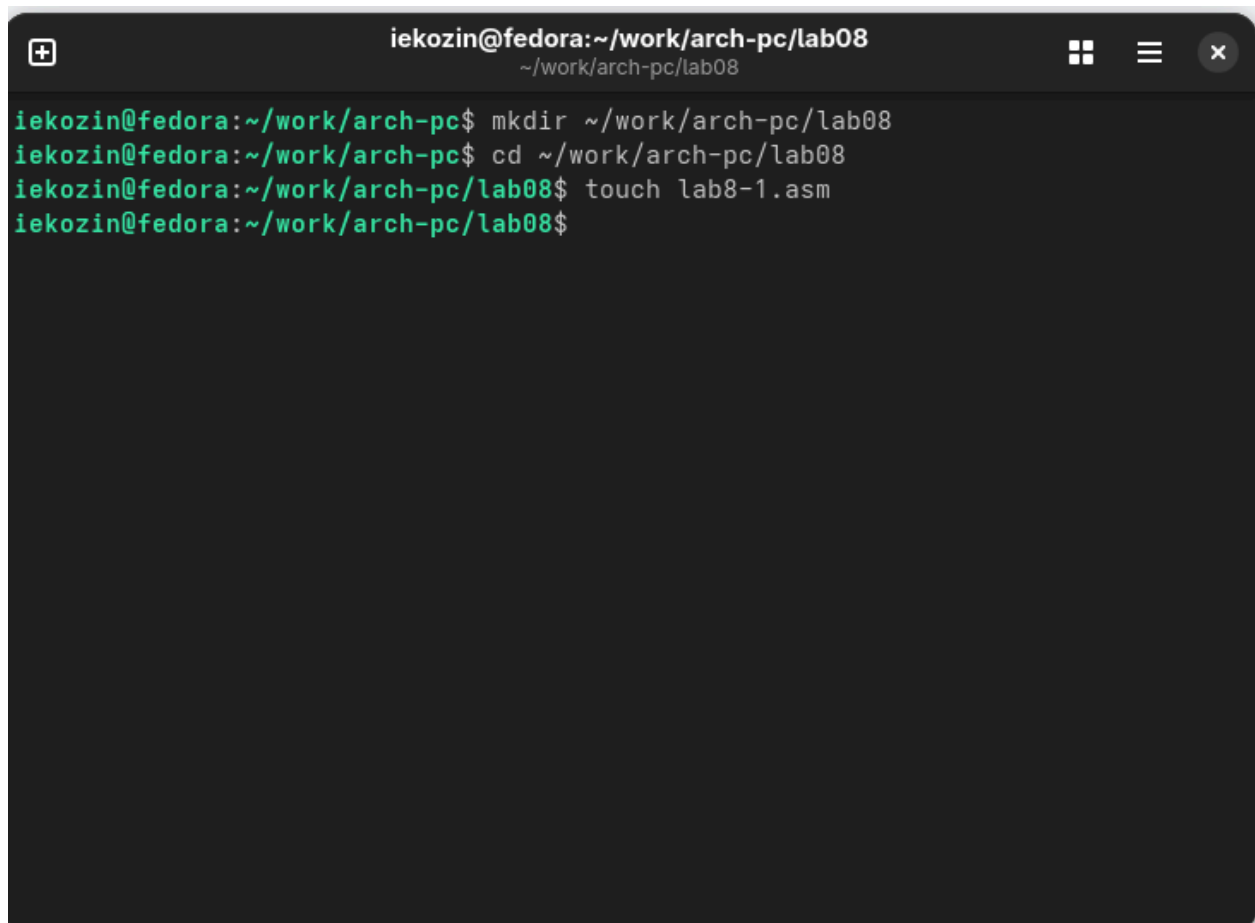
3 Теоретическое введение

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.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы №8 (рис. 1).



```
iekozin@fedora:~/work/arch-pc/lab08
~/work/arch-pc/lab08

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

Рис. 1: Создание каталога

Копирую в созданный файл программу из листинга. (рис. 2).



```
mc [iekozin@fedora]:~/work/arch-pc/lab08 — /usr/bin/mc -P /tmp/mc.pwd.4ueUCc
~/work/arch-pc/lab08

; 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 iprintf ; Вывод значения 'N'
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit
```

Рис. 2: Копирование программы из листинга

Запускаю программу, она показывает работу циклов в NASM (рис. 3).

```
iekozin@fedora:~/work/arch-pc/lab08
~/work/arch-pc/lab08
iekozin@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
iekozin@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
iekozin@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 12
12
11
10
9
8
7
6
5
4
3
2
1
iekozin@fedora:~/work/arch-pc/lab08$
```

Рис. 3: Запуск программы

Заменяю программу изначальную так, что в теле цикла я изменяю значение регистра `ecx` (рис. 4).

```

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 ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`

call quit

```

Рис. 4: Изменение программы

Из-за того, что теперь регистр ecx на каждой итерации уменьшается на 2 значения, количество итераций уменьшается вдвое (рис. 5).

```

iekozin@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
iekozin@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
iekozin@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
7
5
3
1

```

Рис. 5: Запуск измененной программы

Добавляю команды push и pop в программу (рис. 6).

```
mc [iekozin@fedora]~/work/arch-pc/lab08 — /usr/bin/mc -P /tmp/mc.pwd.QhLBvj
~ /work/arch-pc/lab08
SECTION .text
GLOBAL _start
_start:

mov eax, msg1
call sprintf

mov ecx, N
mov edx, 10
call sread

mov eax, N
call atoi
mov [h], eax

mov ecx, [N]

label:
push ecx
sub ecx, 1
mov [N], ecx
mov eax, [N]
call iprintf
pop ecx
loop label
call quit
```

Рис. 6: Добавление push и pop в цикл программы

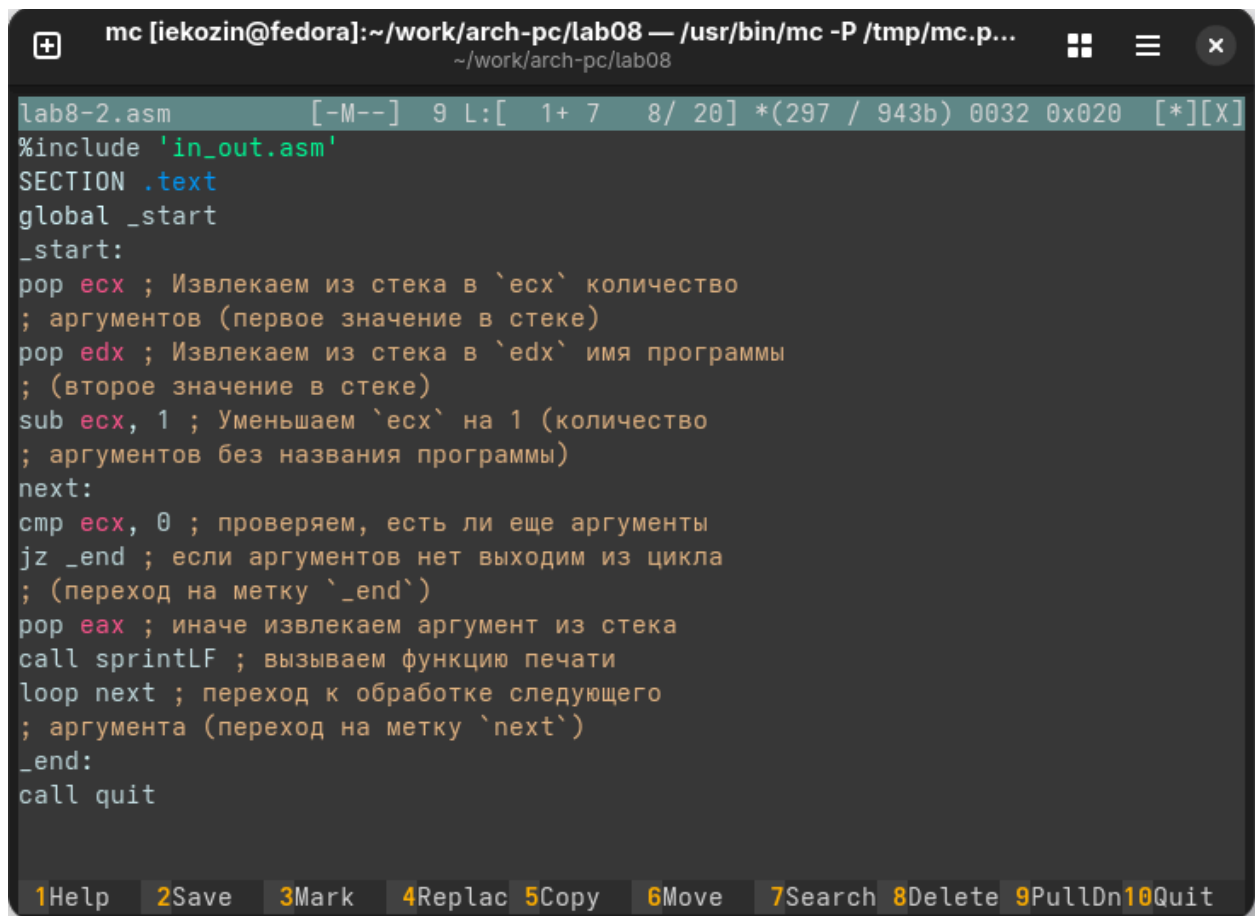
Теперь количество итераций совпадает введенному N, но произошло смещение выводимых чисел на -1 (рис. 7).

```
iekozin@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
iekozin@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
iekozin@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
iekozin@fedora:~/work/arch-pc/lab08$
```

Рис. 7: Запуск измененной программы

4.2 Обработка аргументов командной строки

Создаю новый файл для программы и копирую в него код из следующего листинга (рис. 8).

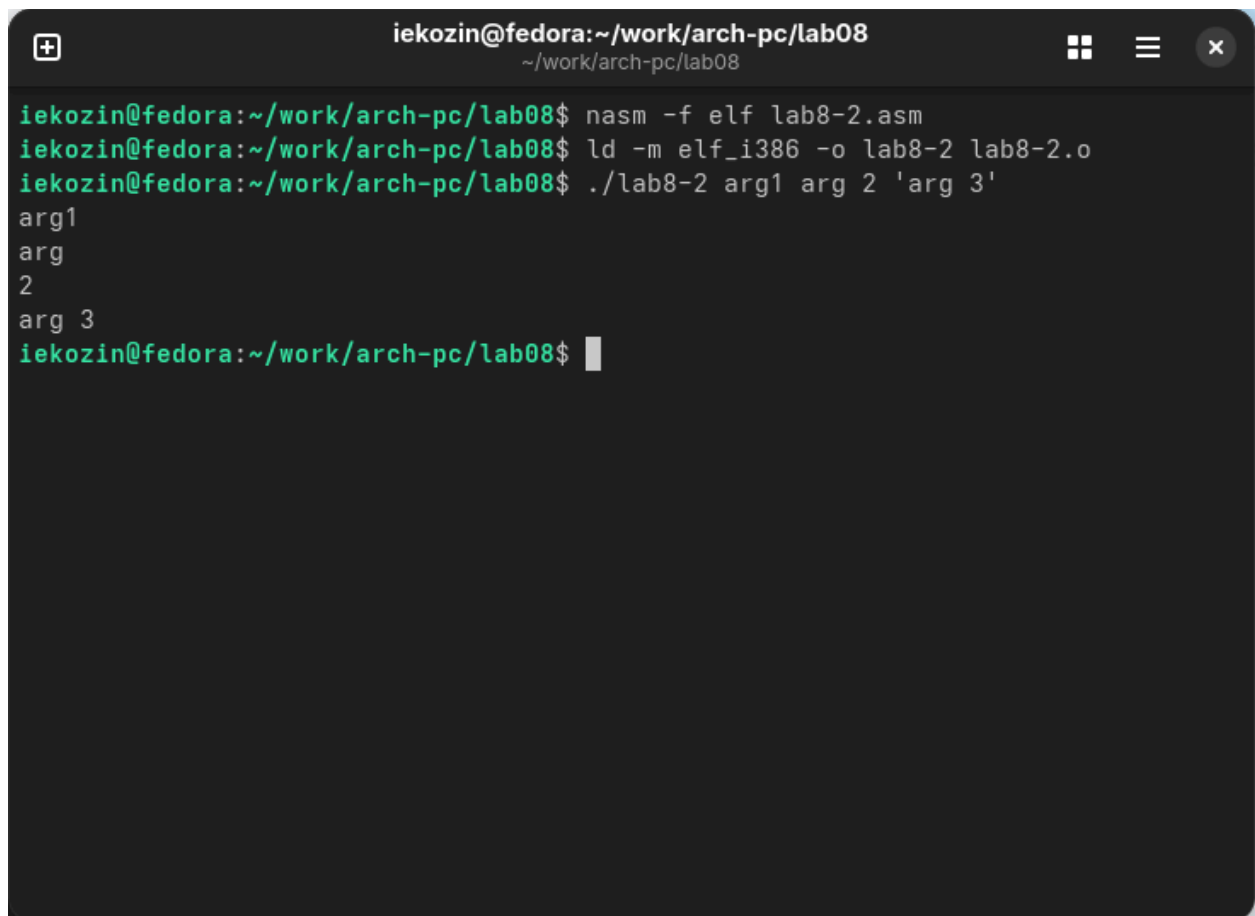


```
lab8-2.asm [-M--] 9 L:[ 1+ 7 8/ 20] *(297 / 943b) 0032 0x020 [*][X]
#include 'in_out.asm'
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx, 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 sprintf ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

1Help 2Save 3Mark 4Replac 5Copy 6Move 7Search 8Delete 9PullDn 10Quit

Рис. 8: Копирование программы из листинга

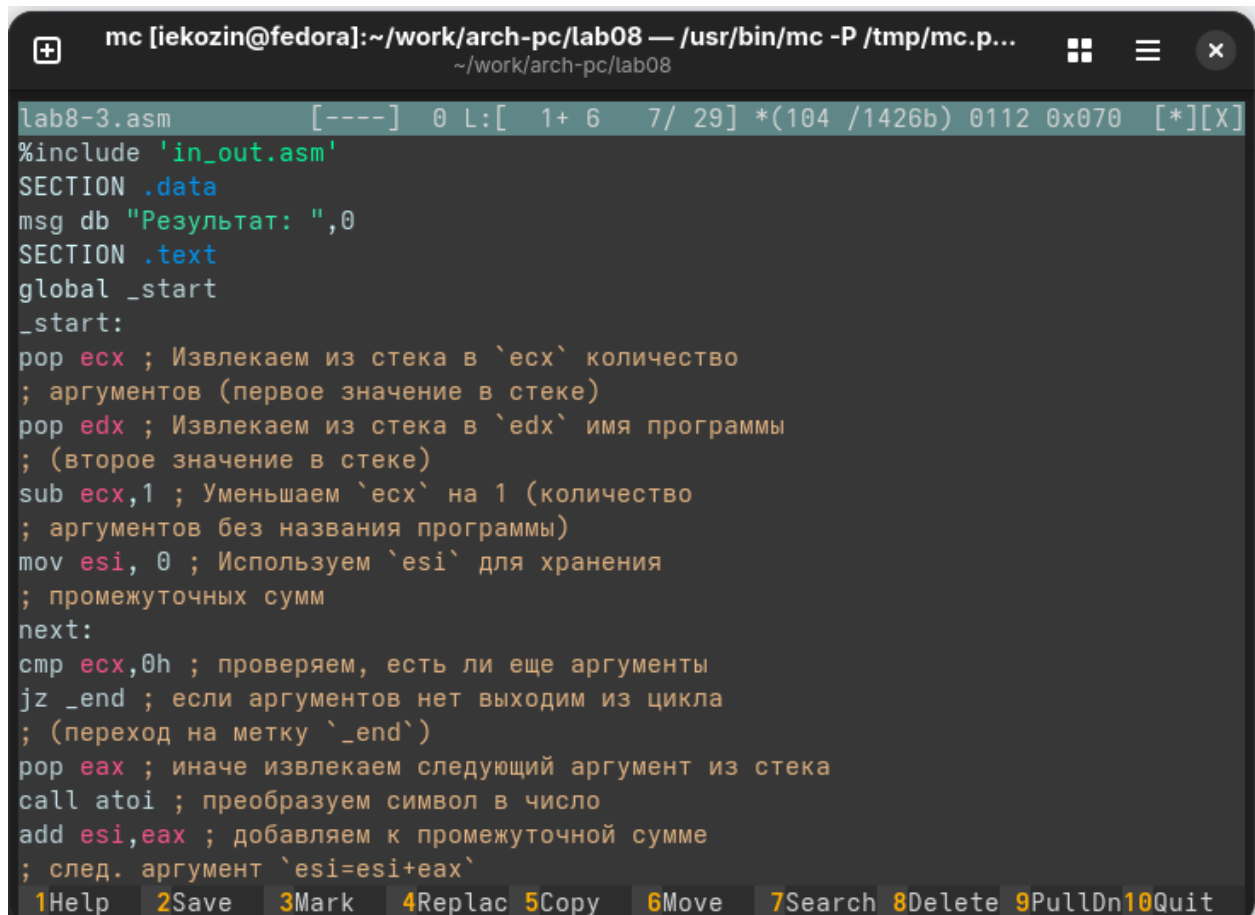
Компилирую программу и запускаю, указав аргументы. Программой было обработано то же количество аргументов, что и было введено (рис. 9).



```
iekozin@fedora:~/work/arch-pc/lab08
~/work/arch-pc/lab08
iekozin@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
iekozin@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
iekozin@fedora:~/work/arch-pc/lab08$ ./lab8-2 arg1 arg 2 'arg 3'
arg1
arg
2
arg 3
iekozin@fedora:~/work/arch-pc/lab08$
```

Рис. 9: Запуск второй программы

Создаю новый файл для программы и копирую в него код из третьего листинга (рис. 10).



```
lab8-3.asm [----] 0 L:[ 1+ 6 7/ 29] *(104 /1426b) 0112 0x070 [*][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`
1Help 2Save 3Mark 4Replac 5Copy 6Move 7Search 8Delete 9PullDn10Quit
```

Рис. 10: Копирование программы из третьего листинга

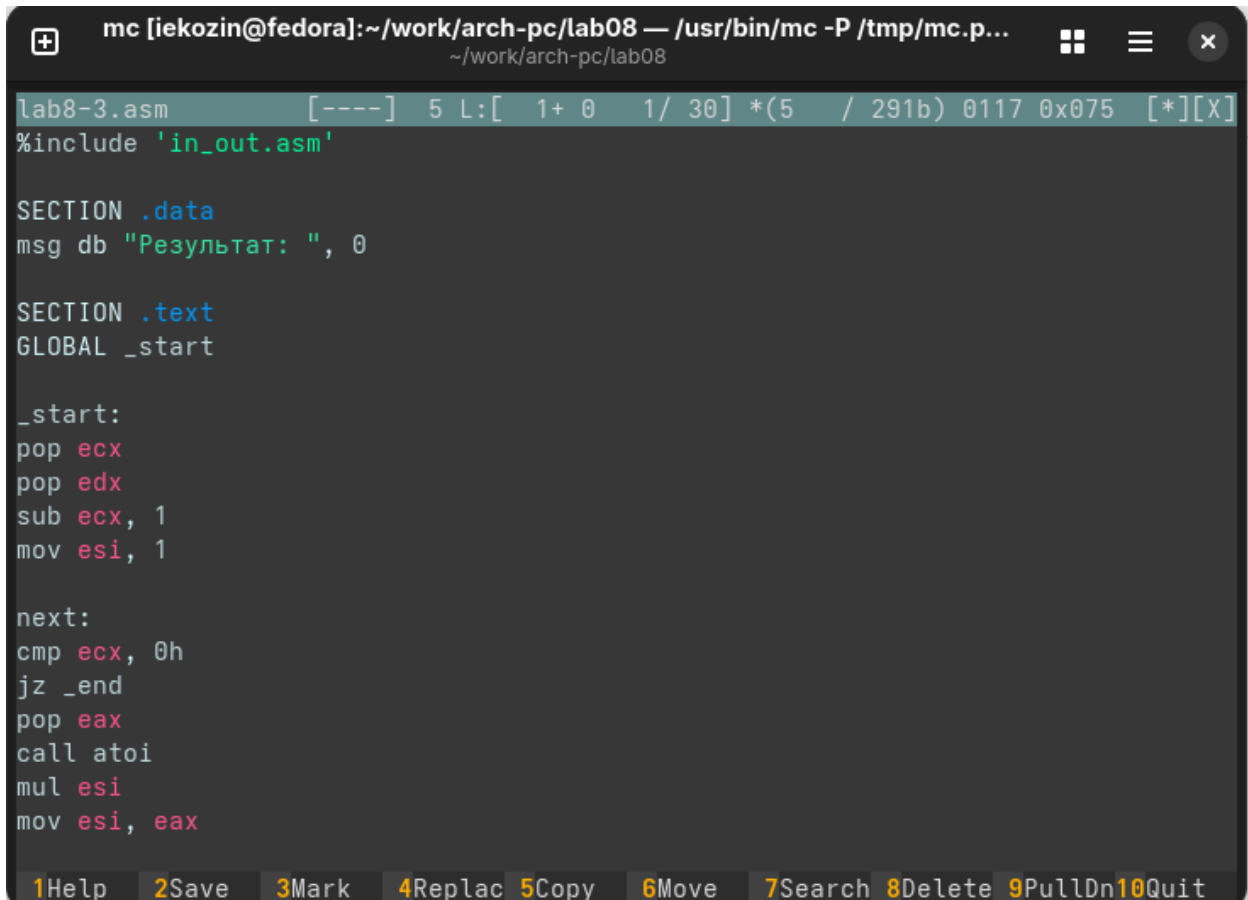
Компилирую программу и запускаю, указав в качестве аргументов некоторые числа, программа их складывает (рис. 11).


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

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

Рис. 11: Запуск третьей программы

Изменяю поведение программы так, чтобы указанные аргументы она умножала, а не складывала (рис. 12).



```
lab8-3.asm [----] 5 L: [ 1+ 0 1/ 30] *(5 / 291b) 0117 0x075 [*][X]
#include 'in_out.asm'

SECTION .data
msg db "Результат: ", 0

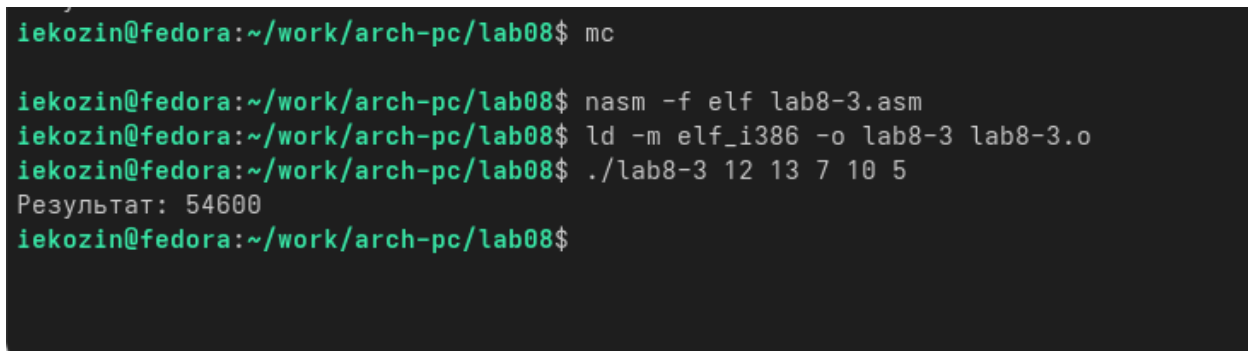
SECTION .text
GLOBAL _start

_start:
pop ecx
pop edx
sub ecx, 1
mov esi, 1

next:
cmp ecx, 0h
jz _end
pop eax
call atoi
mul esi
mov esi, eax
```

Рис. 12: Изменение третьей программы

Программа действительно теперь умножает данные на вход числа (рис. 13).



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

Рис. 13: Запуск измененной третьей программы

4.3 Задание для самостоятельной работы

Пишу программу, которая будет находить сумма значений для функции $f(x) = 2 \cdot (x-1)$, которая совпадает с моим четвертым вариантом (рис. 14).

```
mc [jekozin@fedora]~/work/arch-pc/lab08 — /usr/bin/mc -P /tmp/mc.pwd.hluC26
~work/arch-pc/lab08

SECTION .data
msg_func db "Функция: f(x) = 2*(x-1)", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
; Выводим описание функции
mov eax, msg_func
call sprintf

; Аргументы передаются через стек
; pop ecx = argc, pop edx = адрес возврата (не используется)
pop ecx
pop edx
sub ecx, 1
mov esi, 0
; esi = сумма

next_arg:
cmp ecx, 0
jz print_result

pop eax
call atoi
; преобразуем в число

; f(x) = 2 * (x - 1)
sub eax, 1
shl eax, 1
add esi, eax
; умножаем на 2
; добавляем к сумме

dec ecx
jmp next_arg

print_result:
mov eax, msg_result
call sprintf
mov eax, esi
call sprintf

call quit
```

Рис. 14: Написание программы для самостоятельной работы

Код программы:

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

```
SECTION .data
```

```
msg_func db "Функция: f(x) = 2*(x-1)", 0
```

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

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
    ; Выводим описание функции
```

```
    mov eax, msg_func
```

```
    call sprintf
```

```
    ; Аргументы передаются через стек
```

```
    ; pop ecx = argc, pop edx = адрес возврата (не используется)
```

```
    pop ecx
    pop edx
    sub ecx, 1          ; уменьшаем на 1, т.к. первый аргумент – имя
программы
    mov esi, 0          ; esi = сумма
```

next_arg:

```
    cmp ecx, 0
    jz print_result
```

```
    pop eax             ; берем следующий аргумент
    call atoi           ; преобразуем в число
```

```
    ;  $f(x) = 2 * (x - 1)$ 
    sub eax, 1          ;  $x - 1$ 
    shl eax, 1          ; умножаем на 2
    add esi, eax        ; добавляем к сумме
```

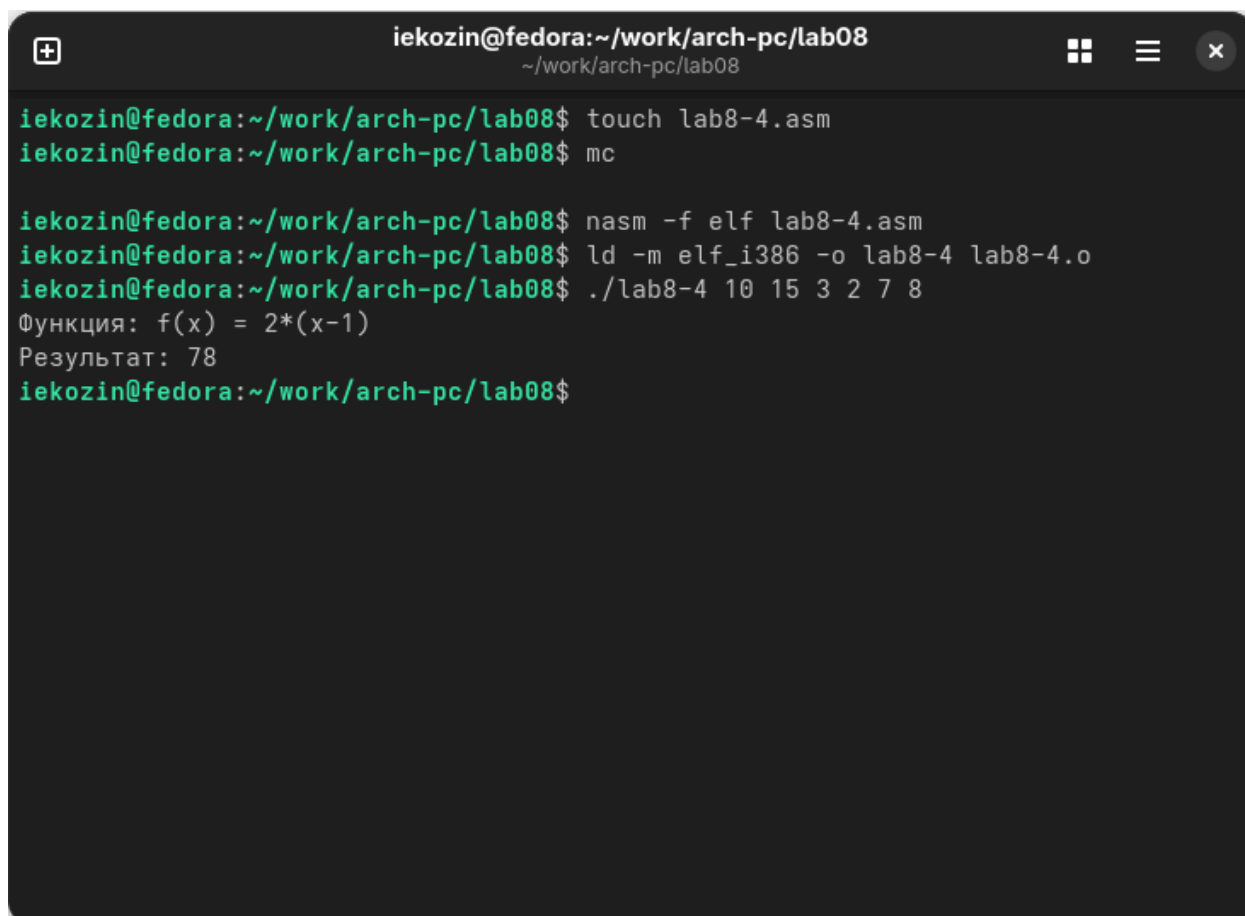
```
    dec ecx
    jmp next_arg
```

print_result:

```
    mov eax, msg_result
    call sprint
    mov eax, esi
    call iprintLF

    call quit
```

Проверяю работу программы, указав в качестве аргумента несколько чисел (рис. 15).

A screenshot of a terminal window with a dark background. The window title is 'iekozin@fedora:~/work/arch-pc/lab08'. The terminal shows the following commands and output:

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

iekozin@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-4.asm
iekozin@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-4 lab8-4.o
iekozin@fedora:~/work/arch-pc/lab08$ ./lab8-4 10 15 3 2 7 8
Функция:  $f(x) = 2 \cdot (x - 1)$ 
Результат: 78
iekozin@fedora:~/work/arch-pc/lab08$
```

Рис. 15: Запуск программы для самостоятельной работы

5 Выводы

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

6 Список литературы

1. Курс на ТУИС
2. Лабораторная работа №8
3. Программирование на языке ассемблера NASM Столяров А. В.