

# 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

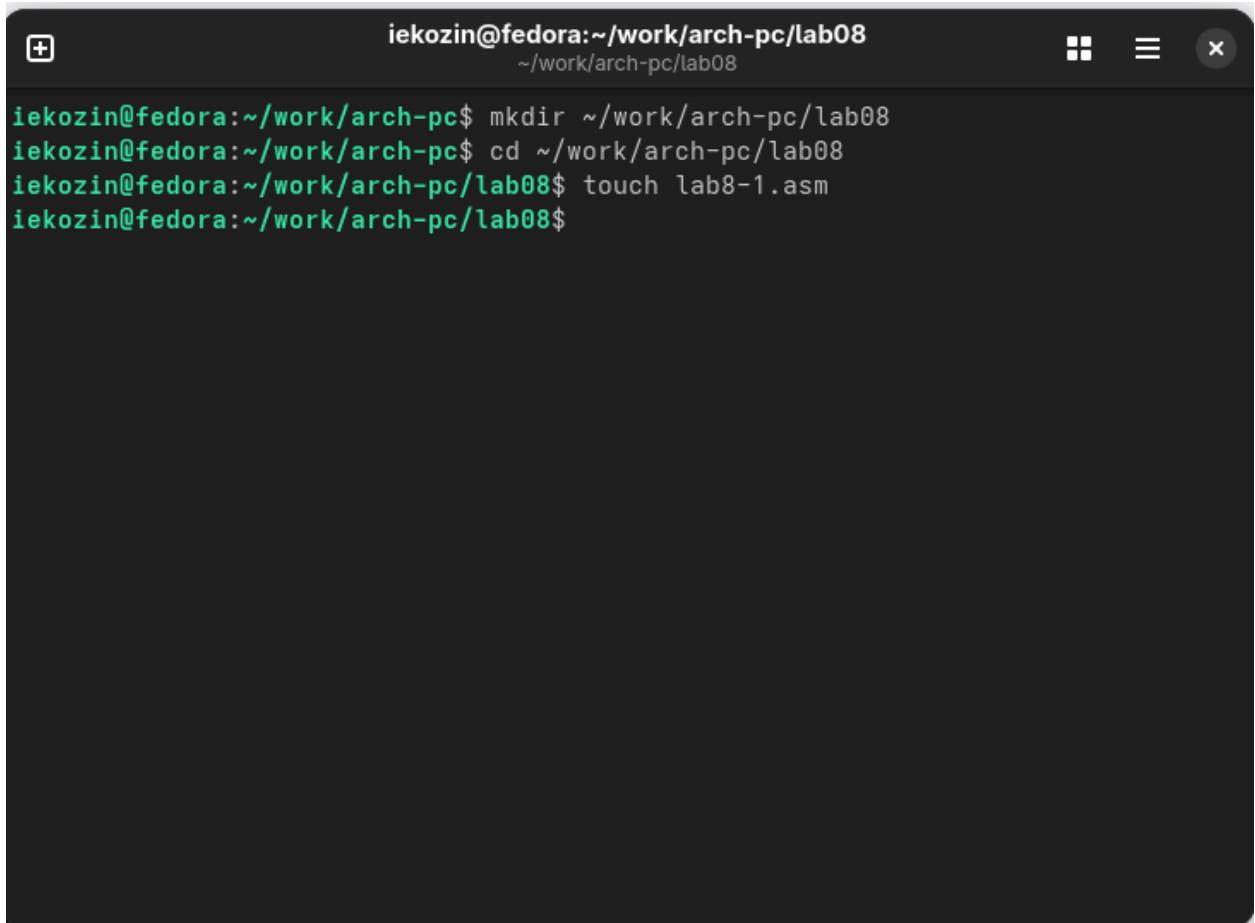
### 3 Теоретическое введение

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

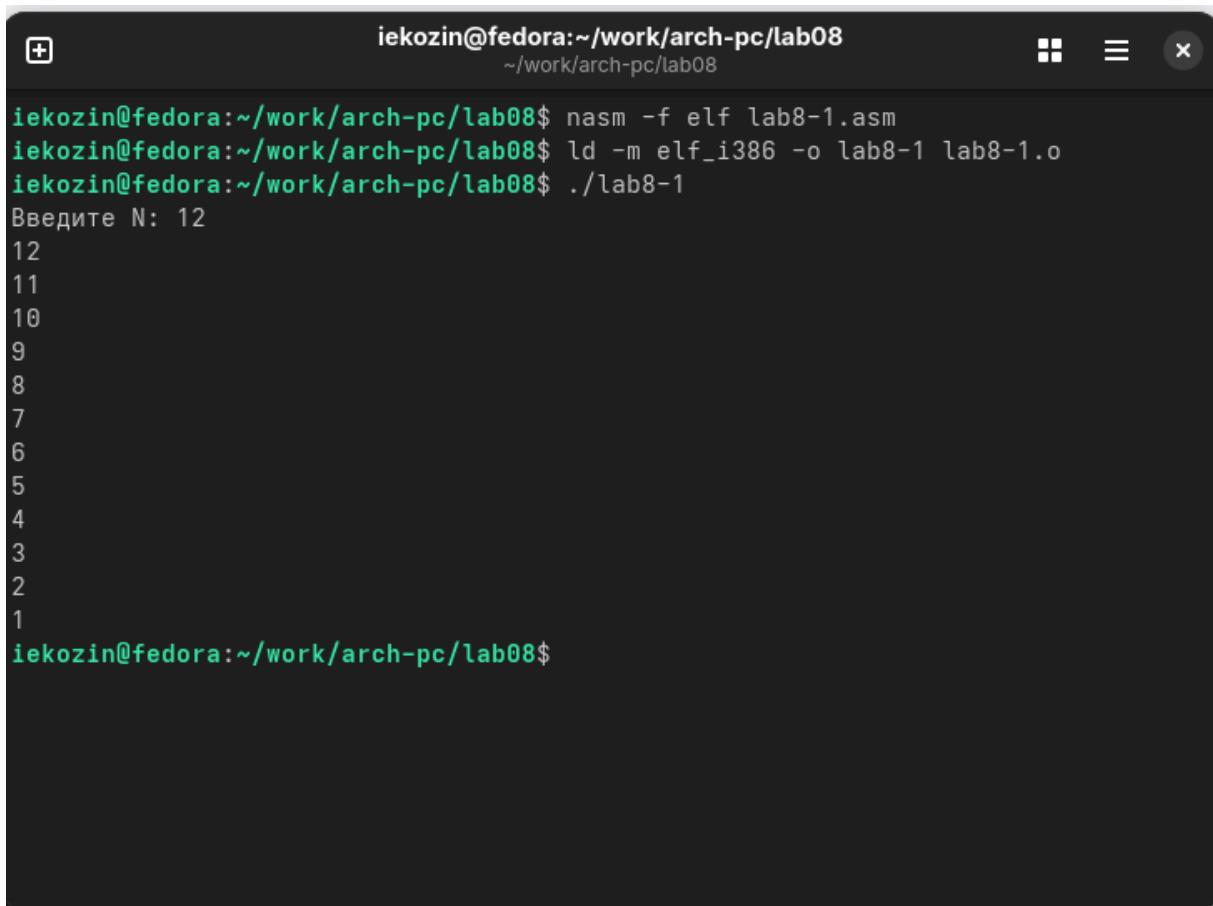
Копирую в созданный файл программу из листинга. (рис. 2).



```
; Программа вывода значений регистра 'есх'
;-----
%include "in_out.asm"
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
SECTON _start
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, `есх=N`
label:
mov [N],ecx
mov eax,[N]
call iprintf ; Вывод значения `N`
loop label ; `есх=есх-1` и если `есх` не `0`
; переход на `label`
call quit
```

Рис. 2: Копирование программы из листинга

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



The screenshot shows a terminal window with the following session:

```
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
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 esx (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).

```
SECTION .text
GLOBAL _start
_start:
    mov eax, msg1
    call sprint

    mov ecx, N
    mov edx, 10
    call sread

    mov ebx, N
    call atoi
    mov [N], ebx

    mov ecx, [N]

label:
    push ecx
    sub ecx, 1
    mov [N], ecx
    mov ebx, [N]
    call lprintf
    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
```

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).

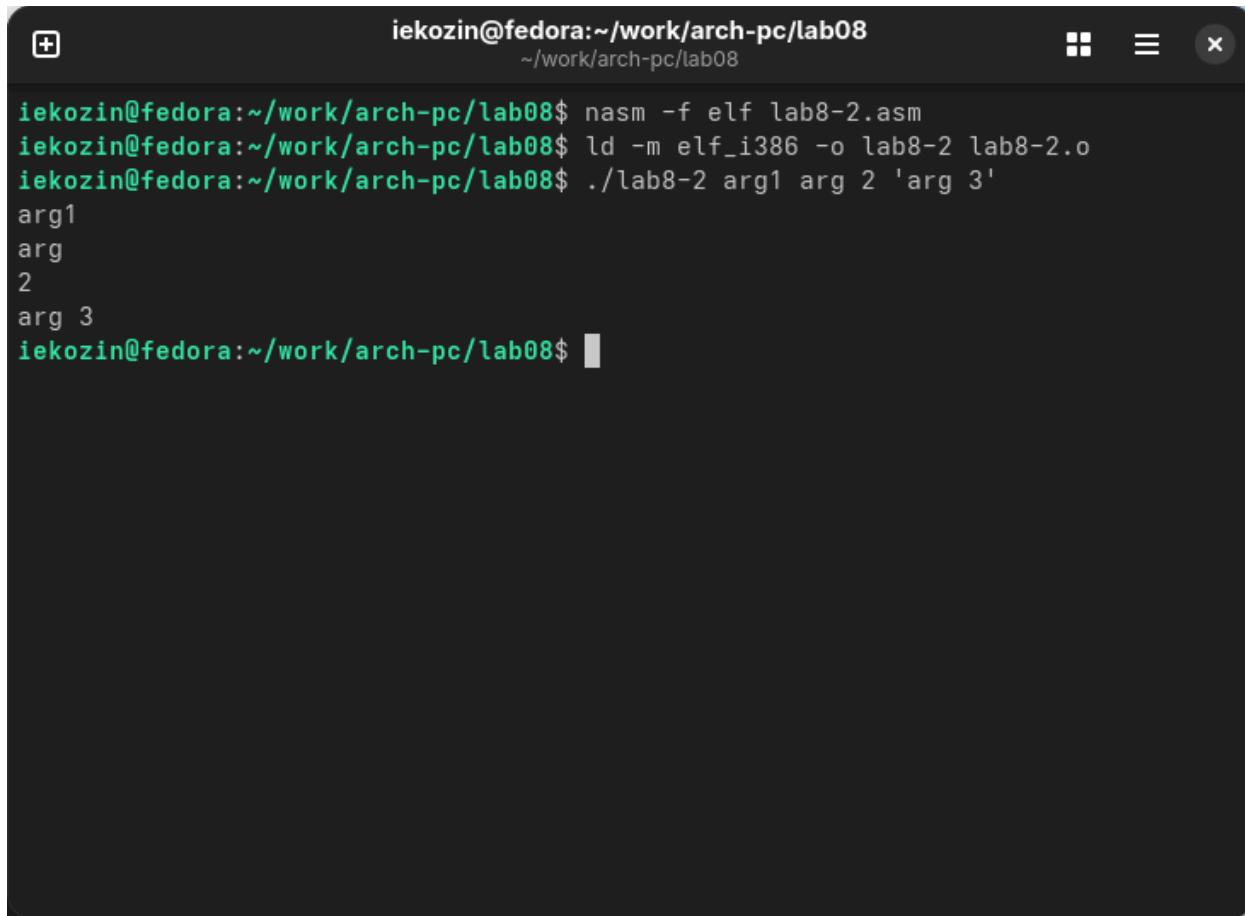
The screenshot shows a terminal window titled 'mc [iekozin@fedora]:~/work/arch-pc/lab08 — /usr/bin/mc -P /tmp/mc.p...'. The command used is 'mc -P /tmp/mc.p...'. The current file is 'lab8-2.asm' located at '~/work/arch-pc/lab08'. The assembly code is as follows:

```
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:
    cmpr 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
    ; аргумента (переход на метку `next`)
_end:
    call quit
```

At the bottom of the window, there is a menu bar with the following options: 1Help, 2Save, 3Mark, 4Replace, 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).



The screenshot shows a terminal window with a dark background and light-colored text. The title bar indicates the user is on a fedora system at the command prompt. The terminal displays the following sequence of commands and their output:

```
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  4Replace  5Copy  6Move  7Search  8Delete  9PullDown  10Quit
```

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).

The screenshot shows the Mars Editor (mc) interface with the assembly code for `lab8-3.asm`. The code includes a section for input/output operations and a main loop that multiplies the value in `ecx` by 5. The assembly instructions are color-coded: `msg`, `SECTION .data`, `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`, and `mov esi, eax`. The menu bar at the bottom includes options like Help, Save, Mark, Replace, Copy, Move, Search, Delete, PullDn, and Quit.

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

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

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

Рис. 12: Изменение третьей программы

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

The terminal window shows the command-line steps to build and run the program. It starts with `mc` to edit the assembly file, followed by `nasm -f elf lab8-3.asm` to assemble it into an ELF file, `ld -m elf_i386 -o lab8-3 lab8-3.o` to link it, and finally `./lab8-3 12 13 7 10 5` to execute the program with the specified arguments. The output is the result of the multiplication: 54600.

```
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 программму, которая будет находить сумму значений для функции  $f(x) = 2^*(x-1)$ , которая совпадает с моим четвертым вариантом (рис. 14).

```

mc [iekozin@fedora]:~/work/arch-pc/lab08 -- /usr/bin/mc -P /tmp/mc.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 sprintLF

; 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 iprintfLF

    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 sprintLF

; 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

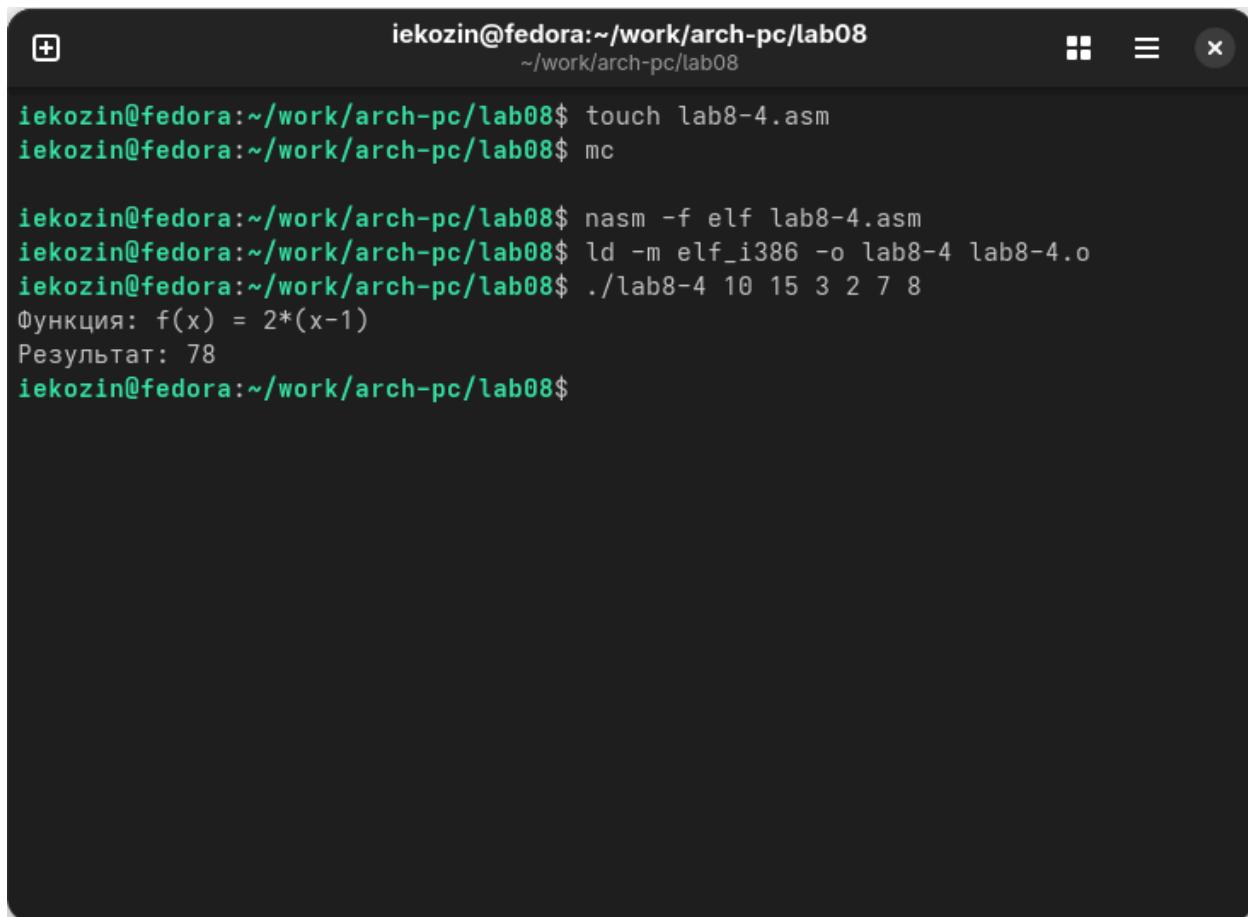
; f(x) = 2 * (x - 1)
sub eax, 1          ; x - 1
shl eax, 1          ; умножаем на 2
add esi, eax        ; добавляем к сумме

dec ecx
jmp next_arg

print_result:
mov eax, msg_result
call sprint
mov eax, esi
call iprintfLF

call quit
```

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



```
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*(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.