

**Лабораторная работа №6. Основы
работы с Midnight Commander
(mc). Структура программы на языке
ассемблера NASM. Системные вызовы в
ОС GNU Linux**

Королёв Иван Андреевич

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	10
4	Задание для самостоятельной работы	16
5	Выводы	19

Список иллюстраций

3.1	MC	10
3.2	lab6	11
3.3	lab6-1.asm	11
3.4	Программа	12
3.5	Программа	13
3.6	in_out.asm	13
3.7	lab6-2.asm	14
3.8	Программа	15
4.1	Программа	16
4.2	Программа	17
4.3	Программа	18
4.4	Программа	18

Список таблиц

1 Цель работы

Приобретение практических навыков работы в Midnight Commander. Освоение инструкций языка ассемблера `mov` и `int`.

2 Теоретическое введение

1. Основы работы с Midnight Commander

Midnight Commander (или просто mc) — это программа, которая позволяет просматривать структуру каталогов и выполнять основные операции по управлению файловой системой, т.е. mc является файловым менеджером. Midnight Commander позволяет сделать работу с файлами более удобной и наглядной. Для активации оболочки Midnight Commander достаточно ввести в командной строке mc и нажать клавишу Enter. В Midnight Commander используются функциональные клавиши F1 — F10, к которым привязаны часто выполняемые операции.

2. Структура программы на языке ассемблера NASM

Программа на языке ассемблера NASM, как правило, состоит из трёх секций: секция кода программы (SECTION .text), секция инициированных (известных во время компиляции) данных (SECTION .data) и секция неинициализированных данных (тех, под которые во время компиляции только отводится память, а значение присваивается в ходе выполнения программы) (SECTION .bss). Таким образом, общая структура программы имеет следующий вид: SECTION .data ; Секция содержит переменные, для ... ; которых задано начальное значение SECTION .bss ; Секция содержит переменные, для ... ; которых не задано начальное значение SECTION .text ; Секция содержит код программы GLOBAL _start _start: ; Точка входа в программу ... ; Текст программы mov eax,1 ; Системный вызов для выхода (sys_exit) mov ebx,0 ; Выход с кодом возврата 0 (без ошибок) int 80h ; Вызов ядра Для объявления инициированных данных в секции .data

используются директивы DB, DW, DD, DQ и DT, которые резервируют память и указывают, какие значения должны храниться в этой памяти: * DB (define byte) — определяет переменную размером в 1 байт; * DW (define word) — определяет переменную размером в 2 байта (слово); * DD (define double word) — определяет переменную размером в 4 байта (двойное слово); * DQ (define quad word) — определяет переменную размером в 8 байт (четырёхбайтное слово); * DT (define ten bytes) — определяет переменную размером в 10 байт. Директивы используются для объявления простых переменных и для объявления массивов. Для определения строк принято использовать директиву DB в связи с особенностями хранения данных в оперативной памяти. Для объявления неинициализированных данных в секции .bss используются директивы resb, resw, resd и другие, которые сообщают ассемблеру, что необходимо зарезервировать заданное количество ячеек памяти.

3. Элементы программирования

- Описание инструкции mov Инструкция языка ассемблера mov предназначена для дублирования данных источника в приёмнике. В общем виде эта инструкция записывается в виде mov dst,src Здесь операнд dst — приёмник, а src — источник. В качестве операнда могут выступать регистры (register), ячейки памяти (memory) и непосредственные значения (const). ВАЖНО! Переслать значение из одной ячейки памяти в другую нельзя, для этого необходимо использовать две инструкции mov: mov eax, x mov y, eax Также необходимо учитывать, что размер операндов приёмника и источника должны совпадать. Использование следующих примеров приведет к ошибке:
 - mov al,1000h — ошибка, попытка записать 2-байтное число в 1-байтный регистр;
 - mov eax,cx — ошибка, размеры операндов не совпадают.
- Описание инструкции int Инструкция языка ассемблера int предназначена для вызова прерывания с указанным номером. В общем виде она запи-

сывается в виде `int n`. Здесь `n` — номер прерывания, принадлежащий диапазону 0–255. При программировании в Linux с использованием вызовов ядра `sys_calls` `n=80h` (принято задавать в шестнадцатеричной системе счисления). После вызова инструкции `int 80h` выполняется системный вызов какой-либо функции ядра Linux. При этом происходит передача управления ядру операционной системы. Чтобы узнать, какую именно системную функцию нужно выполнить, ядро извлекает номер системного вызова из регистра `eax`. Поэтому перед вызовом прерывания необходимо поместить в этот регистр нужный номер. Кроме того, многим системным функциям требуется передавать какие-либо параметры. По принятым в ОС Linux правилам эти параметры помещаются в порядке следования в остальные регистры процессора: `ebx`, `ecx`, `edx`. Если системная функция должна вернуть значение, то она помещает его в регистр `eax`.

- Системные вызовы для обеспечения диалога с пользователем. Простейший диалог с пользователем требует наличия двух функций — вывода текста на экран и ввода текста с клавиатуры. Простейший способ вывести строку на экран — использовать системный вызов `write`. Этот системный вызов имеет номер 4, поэтому перед вызовом инструкции `int` необходимо поместить значение 4 в регистр `eax`. Первым аргументом `write`, помещаемым в регистр `ebx`, задаётся дескриптор файла. Для вывода на экран в качестве дескриптора файла нужно указать 1 (это означает «стандартный вывод», т.е. вывод на экран). Вторым аргументом задаётся адрес выводимой строки (помещаем его в регистр `ecx`, например, инструкцией `mov ecx, msg`). Строка может иметь любую длину. Последним аргументом (т.е. в регистре `edx`) должна задаваться максимальная длина выводимой строки. Для ввода строки с клавиатуры можно использовать аналогичный системный вызов `read`. Его аргументы — такие же, как у вызова `write`, только для «чтения» с клавиатуры используется файловый дескриптор 0 (стандартный ввод). Системный вызов `exit` является обязательным в конце любой программы на языке ас-

семблер. Для обозначения конца программы перед вызовом инструкции `int 80h` необходимо поместить в регистр `eax` значение 1, а в регистр `ebx` код завершения 0.

3 Выполнение лабораторной работы

1. Открываю mc.Перейду в каталог ~/work/arch- pc созданный при выполнении лабораторной работы №53.1

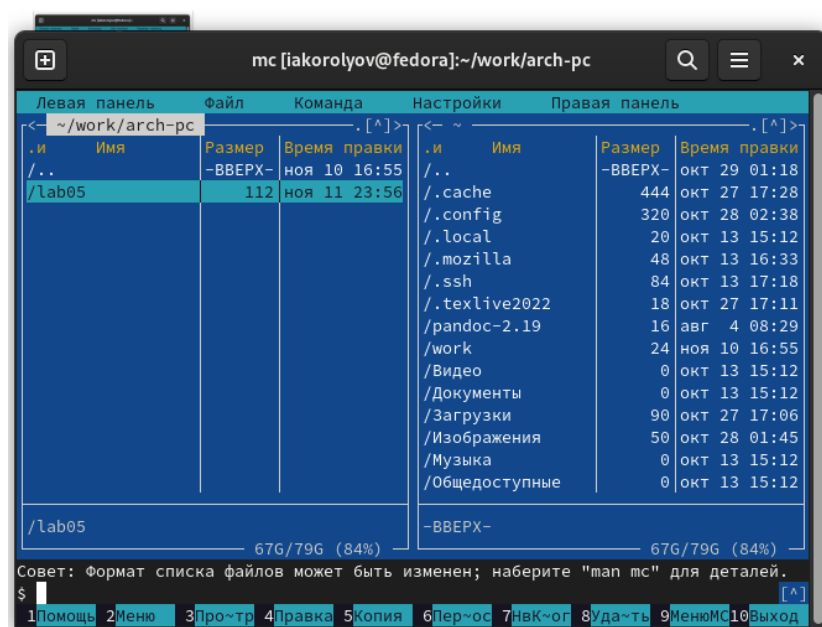


Рис. 3.1: MC

2. С помощью функциональной клавиши F7 создаю папку lab06 и перейду в созданный каталог.3.2

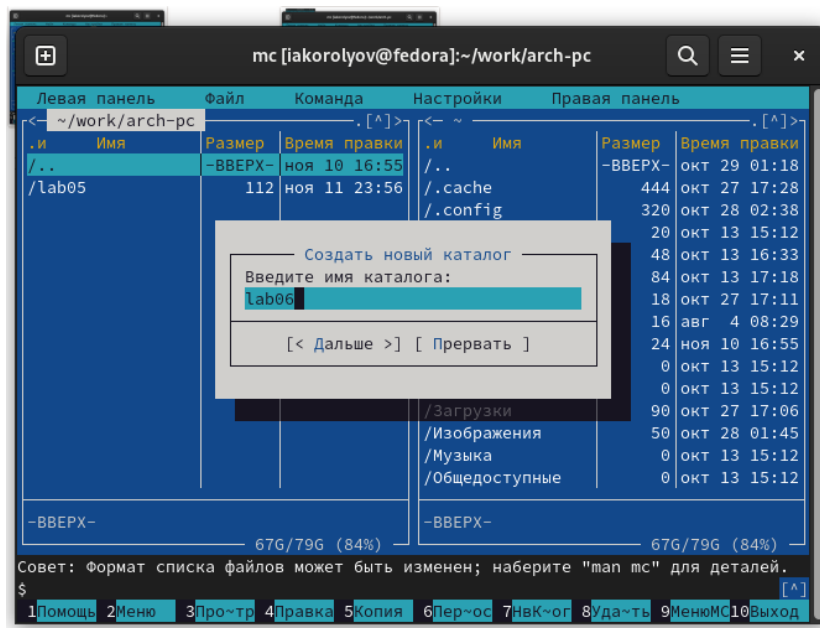


Рис. 3.2: lab6

3. Пользуясь строкой ввода и командой touch создаю файл lab6-1.asm 3.3

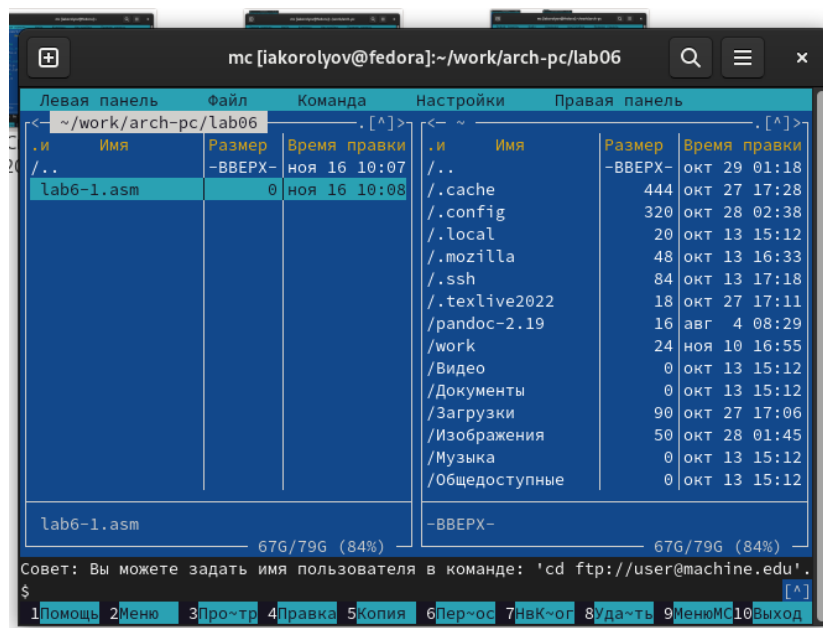
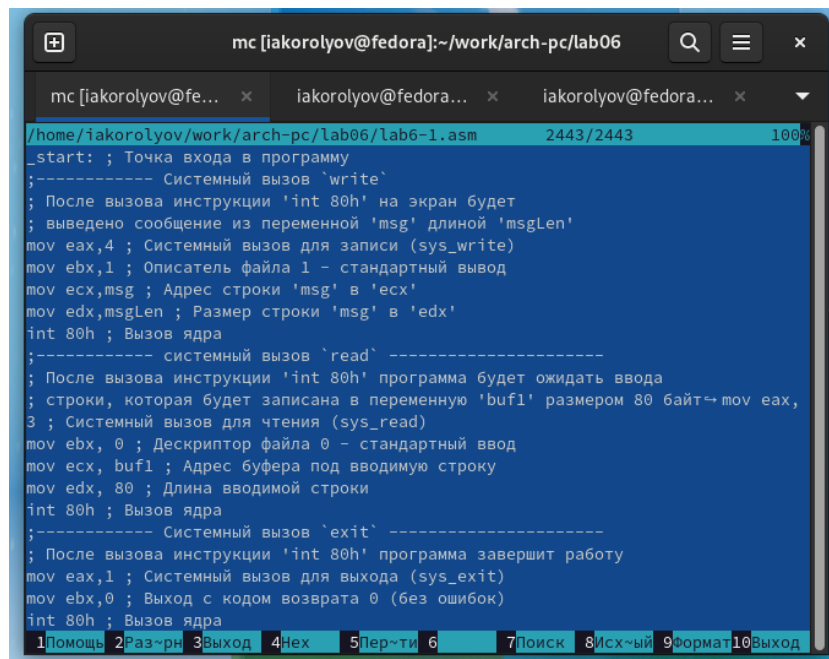


Рис. 3.3: lab6-1.asm

4. С помощью функциональной клавиши F4 открою файл lab6-1.asm. Введу текст программы из листинга 6.1 сохраню изменения и закрою файл. 3.4



```
mc [iakorolyov@fedora]:~/work/arch-pc/lab06
/home/iakorolyov/work/arch-pc/lab06/lab6-1.asm 2443/2443 100%
_start: ; Точка входа в программу
;----- Системный вызов `write`
; После вызова инструкции `int 80h` на экран будет
; выведено сообщение из переменной `msg` длиной `msgLen`
mov eax,4 ; Системный вызов для записи (sys_write)
mov ebx,1 ; Описатель файла 1 - стандартный вывод
mov ecx,msg ; Адрес строки `msg` в `ecx`
mov edx,msgLen ; Размер строки `msg` в `edx`
int 80h ; Вызов ядра
;----- системный вызов `read` -----
; После вызова инструкции `int 80h` программа будет ожидать ввода
; строки, которая будет записана в переменную `buf1` размером 80 байт
mov eax,3 ; Системный вызов для чтения (sys_read)
mov ebx,0 ; Дескриптор файла 0 - стандартный ввод
mov ecx,buf1 ; Адрес буфера под вводимую строку
mov edx,80 ; Длина вводимой строки
int 80h ; Вызов ядра
;----- Системный вызов `exit` -----
; После вызова инструкции `int 80h` программа завершит работу
mov eax,1 ; Системный вызов для выхода (sys_exit)
mov ebx,0 ; Выход с кодом возврата 0 (без ошибок)
int 80h ; Вызов ядра
1Помощь 2Раз-рн 3Выход 4Нех 5Пер-ти 6 7Поиск 8Исх-ый 9Формат10Выход
```

Рис. 3.4: Программа

5. Оттранслирую текст программы lab6-1.asm в объектный файл. Выполню компоновку объектного файла и запущу получившийся исполняемый файл.3.5

```

iakorolyov@fedora:~/work/arch-pc/lab06 — ./lab6-1
[iakorolyov@fedora ~]$ mc
[iakorolyov@fedora lab06]$ ды
bash: ды: команда не найдена...
[iakorolyov@fedora lab06]$ ls
lab6-1.asm
[iakorolyov@fedora lab06]$ nasm -f elf lab6-1.asm
[iakorolyov@fedora lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[iakorolyov@fedora lab06]$ ls
lab6-1  lab6-1.asm  lab6-1.o
[iakorolyov@fedora lab06]$ ./lab6-1
Введите строку:
Korolev Ivan

```

Рис. 3.5: Программа

6. Скачаю файл in_out.asm со страницы курса в ТУИС.Скопирую файл in_out.asm в каталог с файлом lab6-1.asm с помощью функциональной клавиши F5 3.6

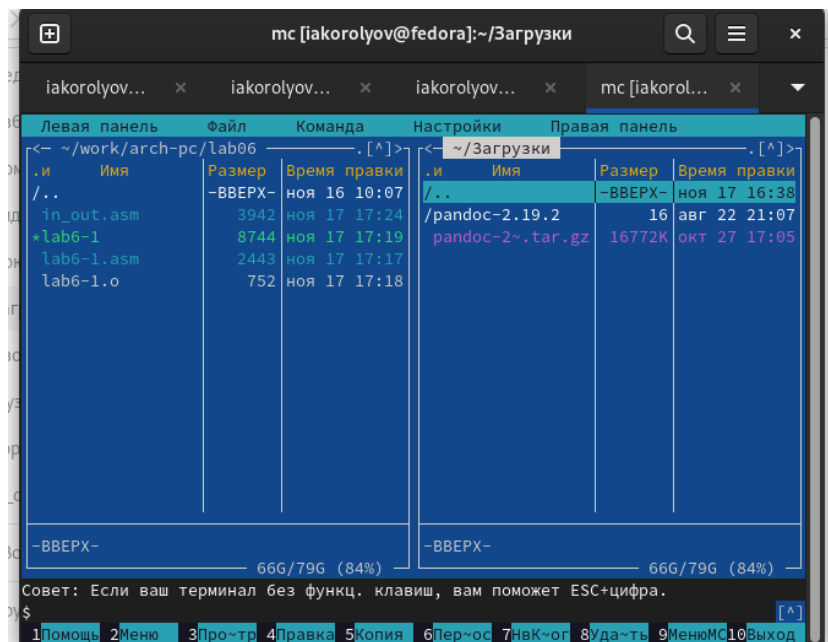
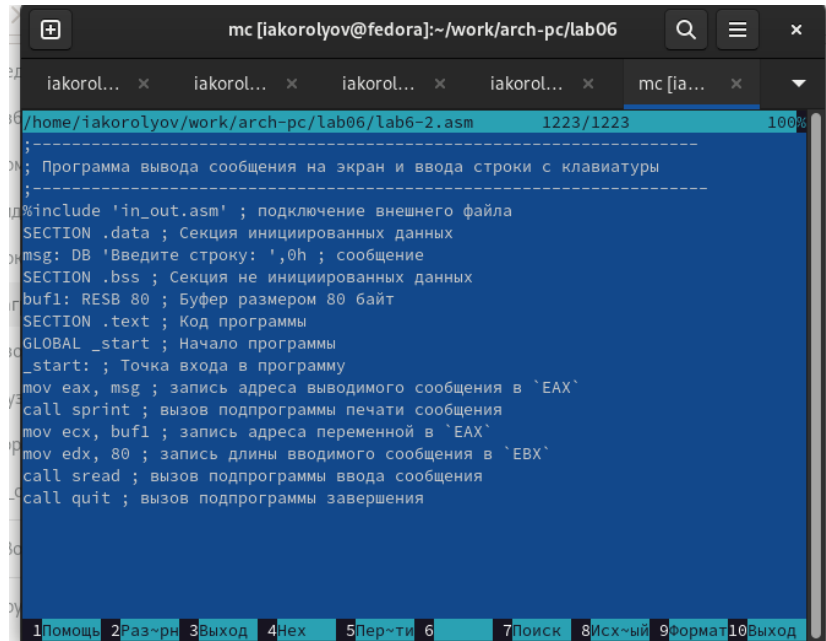


Рис. 3.6: in_out.asm

7. С помощью функциональной клавиши F6 создаю копию файла lab6- 1.asm с именем lab6-2.asm.Выделю файл lab6-1.asm,нажму клавишу F6 , введу имя файла lab6-2.asm и нажму клавишу Enter 3.7

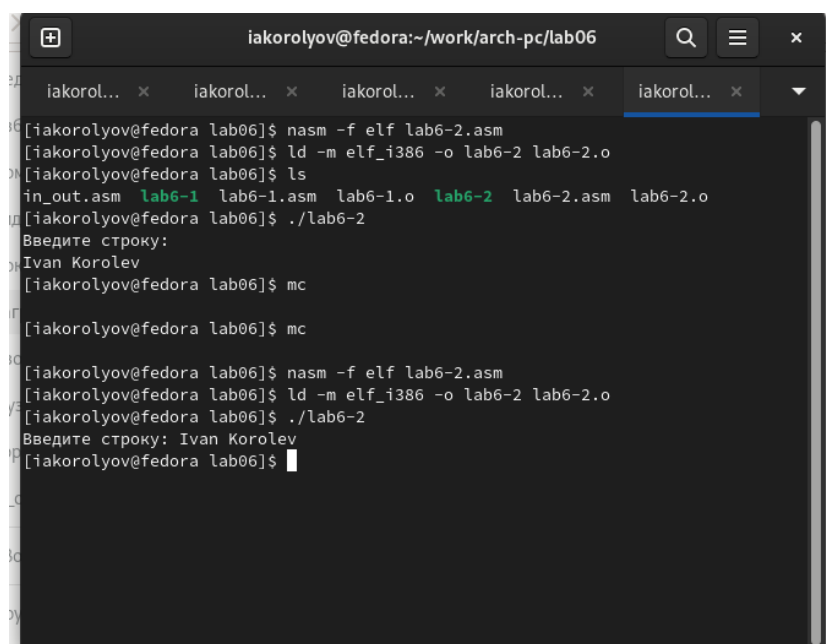


```
mc [iakorolyov@fedora]:~/work/arch-pc/lab06
/home/iakorolyov/work/arch-pc/lab06/lab6-2.asm 1223/1223 100%
;-----
; Программа вывода сообщения на экран и ввода строки с клавиатуры
;-----
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data ; Секция иницированных данных
msg: DB 'Введите строку: ',0h ; сообщение
SECTION .bss ; Секция не иницированных данных
buf1: RESB 80 ; Буфер размером 80 байт
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу
mov eax, msg ; запись адреса выводимого сообщения в `EAX`
call sprint ; вызов подпрограммы печати сообщения
mov ecx, buf1 ; запись адреса переменной в `EAX`
mov edx, 80 ; запись длины вводимого сообщения в `EBX`
call sread ; вызов подпрограммы ввода сообщения
call quit ; вызов подпрограммы завершения

1Помощь 2Раз-рн 3Выход 4Нех 5Пер-ти 6 7Поиск 8Исх-ый 9Формат10Выход
```

Рис. 3.7: lab6-2.asm

8. В файле lab6-2.asm заменю подпрограмму sprintLF на sprint. Создам исполняемый файл и проверю его работу. Происходит переход на новую строку 3.8



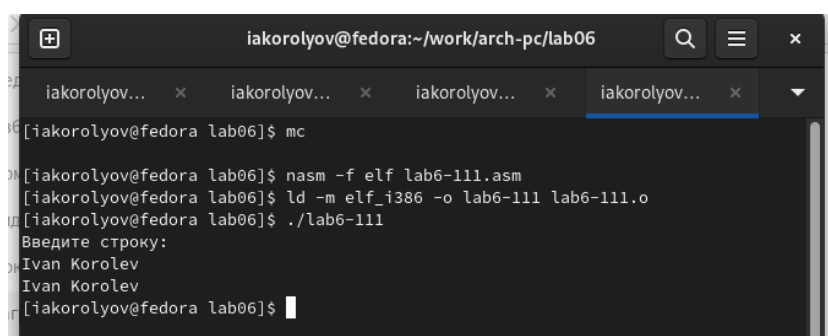
The image shows a terminal window titled 'iakorolyov@fedora:~/work/arch-pc/lab06'. The terminal contains the following commands and output:

```
[iakorolyov@fedora lab06]$ nasm -f elf lab6-2.asm
[iakorolyov@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[iakorolyov@fedora lab06]$ ls
in_out.asm  lab6-1  lab6-1.asm  lab6-1.o  lab6-2  lab6-2.asm  lab6-2.o
[iakorolyov@fedora lab06]$ ./lab6-2
Введите строку:
Ivan Korolev
[iakorolyov@fedora lab06]$ mc
[iakorolyov@fedora lab06]$ mc
[iakorolyov@fedora lab06]$ nasm -f elf lab6-2.asm
[iakorolyov@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[iakorolyov@fedora lab06]$ ./lab6-2
Введите строку: Ivan Korolev
[iakorolyov@fedora lab06]$
```

Рис. 3.8: Программа

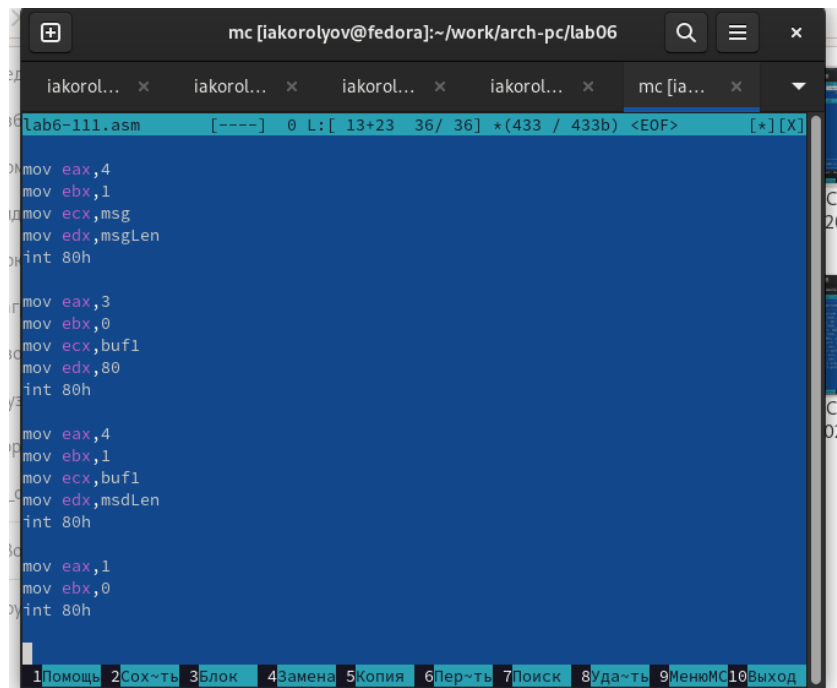
4 Задание для самостоятельной работы

1. Создаю копию файла lab6-1.asm. Внесу изменения в программу (без использования внешнего файла in_out.asm), так чтобы она работала по алгоритму.4.1,4.2



```
iakorolyov@fedora:~/work/arch-pc/lab06
[iakorolyov@fedora lab06]$ mc
[iakorolyov@fedora lab06]$ nasm -f elf lab6-111.asm
[iakorolyov@fedora lab06]$ ld -m elf_i386 -o lab6-111 lab6-111.o
[iakorolyov@fedora lab06]$ ./lab6-111
Введите строку:
Ivan Korolev
Ivan Korolev
[iakorolyov@fedora lab06]$
```

Рис. 4.1: Программа



```
mc [iakorolyov@fedora]:~/work/arch-pc/lab06
lab6-111.asm [----] 0 L: [ 13+23 36/ 36] *(433 / 433b) <EOF> [*] [X]

mov eax,4
mov ebx,1
mov ecx,msg
mov edx,msgLen
int 80h

mov eax,3
mov ebx,0
mov ecx,buf1
mov edx,80
int 80h

mov eax,4
mov ebx,1
mov ecx,buf1
mov edx,msgLen
int 80h

mov eax,1
mov ebx,0
int 80h

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Удалить 9МенюMC10Выход
```

Рис. 4.2: Программа

2. Создаю копию файла lab6-2.asm. Внесу изменения в программу с использования внешнего файла in_out.asm, так чтобы она работала по алгоритму.4.3,4.4

```

iakorolyov@fedora:~/work/arch-pc/lab06
[iakorolyov@fedora lab06]$ nasm -f elf lab6-22.asm
[iakorolyov@fedora lab06]$ ld -m elf_i386 -o lab6-22 lab6-22.o
[iakorolyov@fedora lab06]$ ./lab6-22
Введите строку:
Ivan Korolev
[iakorolyov@fedora lab06]$ ls
in_out.asm  lab6-111  lab6-11.asm  lab6-1.asm  lab6-22  lab6-2.asm
lab6-1      lab6-111.asm  lab6-11.asm.save  lab6-1.o  lab6-22.asm  lab6-2.o
lab6-11     lab6-111.o  lab6-11.o  lab6-2     lab6-22.o
[iakorolyov@fedora lab06]$ mc
[iakorolyov@fedora lab06]$ nasm -f elf lab6-22.asm
[iakorolyov@fedora lab06]$ ld -m elf_i386 -o lab6-22 lab6-22.o
[iakorolyov@fedora lab06]$ ./lab6-22
Введите строку:
Ivan K0role
Ошибка сегментирования (стек памяти сброшен на диск)
[iakorolyov@fedora lab06]$ mc
[iakorolyov@fedora lab06]$ nasm -f elf lab6-22.asm
[iakorolyov@fedora lab06]$ ld -m elf_i386 -o lab6-22 lab6-22.o
[iakorolyov@fedora lab06]$ ./lab6-22
Введите строку:
Ivan Korolev
Ivan Korolev
TTTTTTTT[iakorolyov@fedora lab06]$

```

Рис. 4.3: Программа

```

mc [iakorolyov@fedora]:~/work/arch-pc/lab06
lab6-22.asm  [----] 11 L: [ 1+17 18/ 21] *(1134/1284b) 0032 0x020 [*][X]
; Программа вывода сообщения на экран и ввода строки с клавиатуры
;-----
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data ; Секция инициализированных данных
msg: DB 'Введите строку: ',0h ; сообщение
msd: DB 'TTTTTTTTTTTTTTTTTT',0h
SECTION .bss ; Секция не инициализированных данных
buf1: RESB 80 ; Буфер размером 80 байт
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу
mov eax, msg ; запись адреса выводимого сообщения в 'EAX'
call sprintf ; вызов подпрограммы печати сообщения
mov ecx, msd ; запись адреса переменной в 'EAX'
mov edx, 80 ; запись длины вводимого сообщения в 'EBX'
call sread
mov eax,msd ; вызов подпрограммы ввода сообщения
call sprint
call quit ; вызов подпрограммы завершения

```

Рис. 4.4: Программа

5 Выводы

Я приобрел практические навыки работы в Midnight Commander. Освоение инструкций языка ассемблера `mov` и `int`.