РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ
ФАКУЛЬТЕТ ФИЗИКО-МАТЕМАТИЧЕСКИХ И ЕСТЕСТВЕННЫХ НАУК

Лабораторная работа №.5

Основы работы с Midnight Commander (mc). Структура программы на языке ассемблера NASM. Системные вызовы в ОС GNU Linux.

Левищева Анастасия Петровна, НКАбд-02-25

Содержание

Список иллюстраций……...3
Список таблиц……4
Цель работы……5
Задание……6
1. Теоретическое введение……7
1.1. Основы работы с Midnight Commander……...7
1.2. Структура программы на языке ассемблера NASM…….8
1.3. Элементы программирования……...9
1.3.1. Описание инструкции mov……...9
1.3.2. Описание инструкции int……10
1.3.3. Системные вызовы для обеспечения диалога с пользователем……10
1.4. Подключение внешнего файла in_out.asm……11
2. Выполнение лабораторной работы…….…...12
Выводы……24
Список литературы…….25

Список иллюстраций

Рис.1.Midnight Commander……..……..12

Рис.2.Каталог ~/work/arch-pc……..……..12

Рис.3.1.Папка lab05……..……..13

Рис.3.2.Каталог lab05……..…13

Рис.4.Файл lab5-1.asm……...……14

Рис.5.Редактирование……14

Рис.6.Текст программы…….……15

Рис.7.Проверка……...…15

Рис.8.Запуск файла lab5-1…….……16

Рис.9.Загрузка файла…….……16

Рис.10.1.Панель mc……16

Рис.10.2.Копируем файл……17

Рис.11.Копия lab5-1.asm……17

Рис.12.1.Редактирование lab5-2.asm……18

Рис.12.2.Проверка lab5-2.asm……...……18

Рис.13.Замена sprintLF на sprint……...……18

Рис.14.1.Файл lab5-3.asm……..…20

Рис.14.2.Редактируем lab5-3.asm…….……21

Рис.14.3.Проверка lab5-3.asm……...……21

Рис.15.Проверка lab5-3.asm(1) ……22

Рис.16.1.Файл lab5-4.asm……..22

Рис.16.2. Редактируем lab5-4.asm…….……23

Рис.16.3.Проверка lab5-4.asm……...……23

Рис.17. Проверка lab5-4.asm(1) ……...……23

Список таблиц

Таблица 5.1. Функциональные клавиши Midnight Commander……7

Цель работы

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

Задание

1. Создайте копию файла lab5-1.asm. Внесите изменения в программу (без использования внешнего файла in_out.asm), так чтобы она работала по следующему алгоритму:

- вывести приглашение типа “Введите строку:”;
- ввести строку с клавиатуры;
- вывести введённую строку на экран.

2. Получите исполняемый файл и проверьте его работу. На приглашение ввести строку введите свою фамилию.

3. Создайте копию файла lab5-2.asm. Исправьте текст программы с использование подпрограмм из внешнего файла in_out.asm, так чтобы она работала по следующему алгоритму:

- вывести приглашение типа “Введите строку:”;
- ввести строку с клавиатуры;
- вывести введённую строку на экран.

4. Создайте исполняемый файл и проверьте его работу.

1. Теоретическое введение

1.1. 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. 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 (табл. 5.1).

Таблица 5.1. Функциональные клавиши Midnight Commander

Функцио-нальные клавиши	Выполняемое действие
F1	вызов контекстно-зависимой подсказки
F2	вызов меню, созданного пользователем
F3	просмотр файла, на который указывает подсветка в активной панели
F4	вызов встроенного редактора для файла, на который указывает подсветка в активной панели
F5	копирование файла или группы отмеченных файлов из каталога, отображаемого в активной панели, в каталог, отображаемый на второй панели
F6	перенос файла или группы отмеченных файлов из каталога, отображаемого в активной панели, в каталог, отображаемый на второй панели
F7	создание подкаталога в каталоге, отображаемом в активной панели
F8	удаление файла (подкаталога) или группы отмеченных файлов
F9	вызов основного меню программы
F10	выход из программы

Следующие комбинации клавиш облегчают работу с Midnight Commander:

• Tab используется для переключений между панелями;

• ↑ и ↓ используется для навигации, Enter для входа в каталог или открытия файла (если в файле расширений mc.ext заданы правила связи определённых расширений файлов с инструментами их запуска или обработки);

• Ctrl + u (или через меню Команда > Переставить панели) меняет местами содержимое правой и левой панелей;
• Ctrl + o (или через меню Команда > Отключить панели) скрывает или возвращает панели Midnight Commander, за которыми доступен для работы командный интерпретатор оболочки и выводимая туда информация.
• Ctrl + x + d (или через меню Команда > Сравнить каталоги) позволяет сравнить содержимое каталогов, отображаемых на левой и правой панелях.

1.2. 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 размеров 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 (учетверённое 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. Синтаксис директив определения данных следующий:

```
<имя> DB <операнд> [, <операнд>] [, <операнд>]
```

Для объявления неинициированных данных в секции .bss используются директивы resb, resw, resd и другие, которые сообщают ассемблеру, что необходимо зарезервировать заданное количество ячеек памяти.

1.3. Элементы программирования

1.3.1. 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 то, что размер операндов приемника и источника должны совпадать.

1.3.2. Описание инструкции int

Инструкция языка ассемблера intпредназначена для вызова прерывания с указанным номером. В общем виде она записывается 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).
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.

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

1.4. Подключение внешнего файла in_out.asm

Для упрощения написания программ часто встречающиеся одинаковые участки кода (такие как, например, вывод строки на экран или выход их программы) можно оформить в виде подпрограмм и сохранить в отдельные файлы, а во всех нужных местах поставить вызов нужной подпрограммы. Это позволяет сделать основную программу более удобной для написания и чтения.
NASM позволяет подключать внешние файлы с помощью директивы %include, которая предписывает ассемблеру заменить эту директиву содержимым файла. Подключаемые файлы также написаны на языке ассемблера. Важно отметить, что директива %include в тексте программы должна стоять раньше, чем встречаются вызовы подпрограмм из подключаемого файла. Для вызова подпрограммы из внешнего файла используется инструкция call, которая имеет следующий вид

call <function>

где function имя подпрограммы.
Для выполнения лабораторных работ используется файл in_out.asm, который содержит следующие подпрограммы:
• slen – вычисление длины строки (используется в подпрограммах печати сообщения для определения количества выводимых байтов);
• sprint – вывод сообщения на экран, перед вызовом sprint в регистр eax необходимо записать выводимое сообщение (mov eax,);
• sprintLF – работает аналогично sprint, но при выводе на экран добавляет к сообщению символ перевода строки;
• sread – ввод сообщения с клавиатуры, перед вызовом sread в регистр eax необходимо записать адрес переменной, в которую введенное сообщение буд записано (mov eax,), в регистр ebx – длину вводимой строки (mov ebx,);
• iprint – вывод на экран чисел в формате ASCII, перед вызовом iprint в регистр eax необходимо записать выводимое число (mov eax,);
• iprintLF – работает аналогично iprint, но при выводе на экран после числа добавляет к символ перевода строки;
• atoi – функция преобразует ascii-код символа в целое число и записыает результат в регистр eax, перед вызовом atoi в регистр eax необходимо записать число (mov eax,);
• quit – завершение программы.

2. Выполнение лабораторной работы

1. Откроем Midnight Commander:

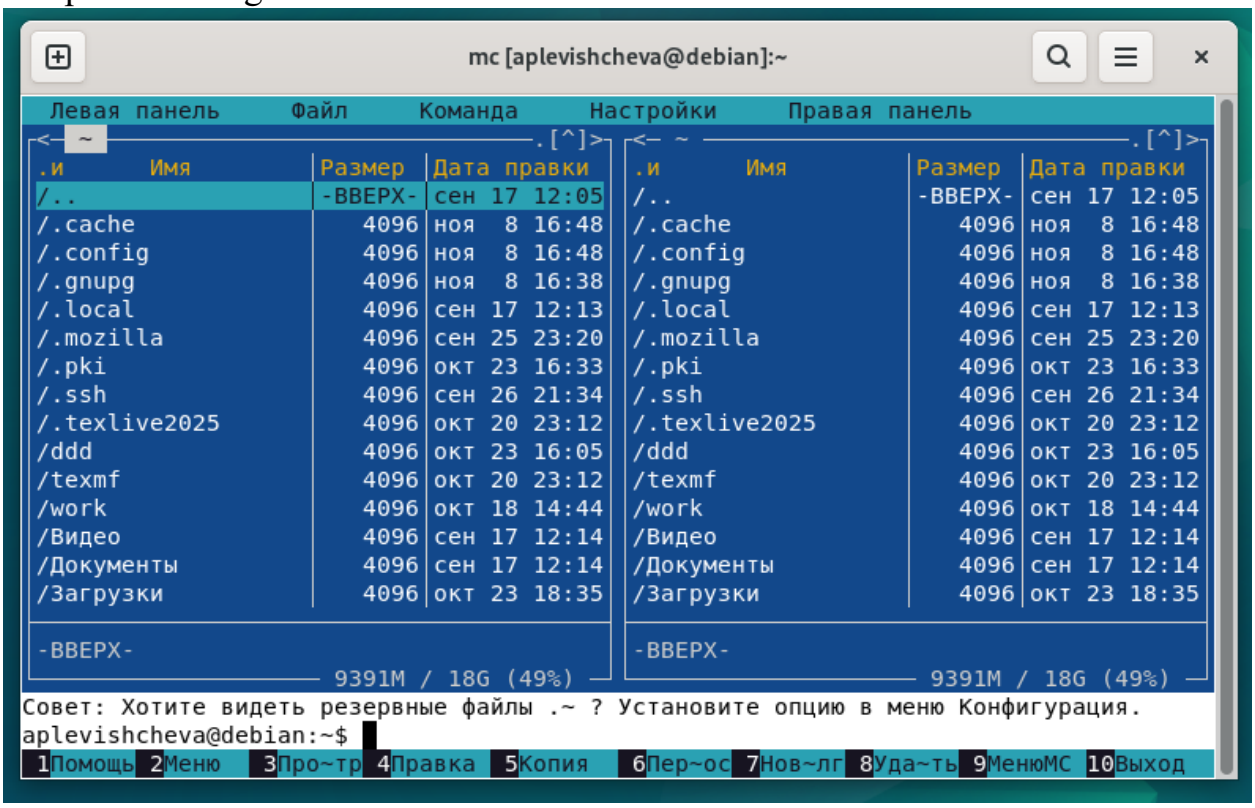

Рис.1.Midnight Commander

2. Прейдем в каталог ~/work/arch-pc, созданный во время выполнения лабораторной работы №4, с помощью клавиш ↑, ↓ и Enter:

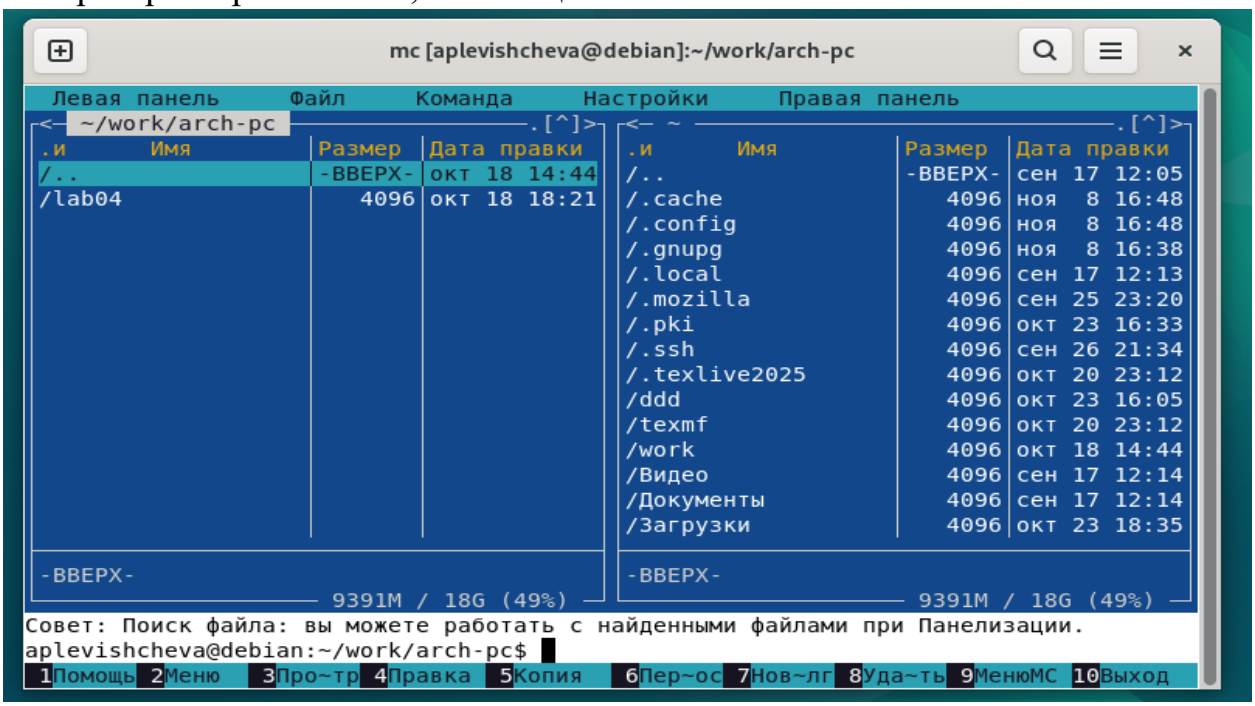

Рис.2.Каталог ~/work/arch-pc

3. С помощью функциональной клавиши F7 создадим папку lab05:

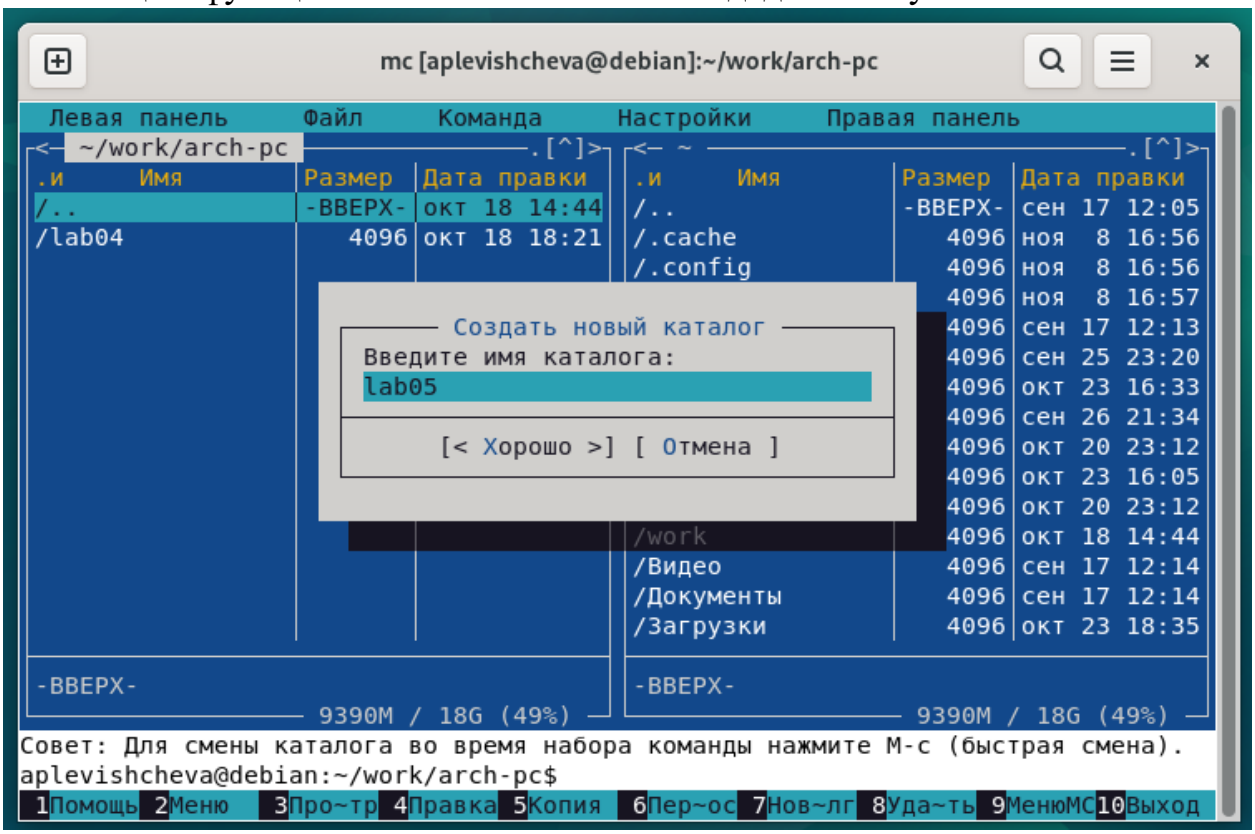

Рис.3.1.Папка lab05

Перейдем в созданный каталог:

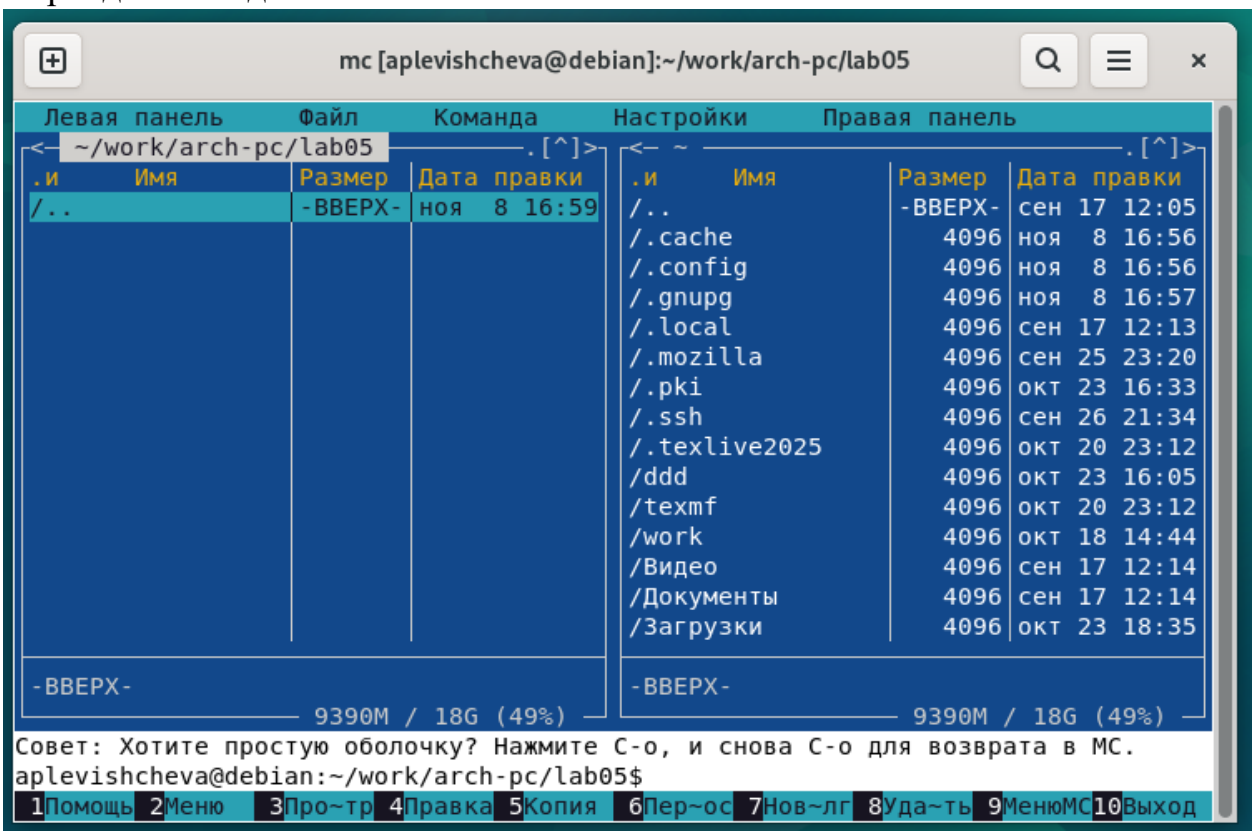

Рис.3.2.Каталог lab05

4. Пользуясь строкой ввода и командой touch, создадим файл lab5-1.asm:

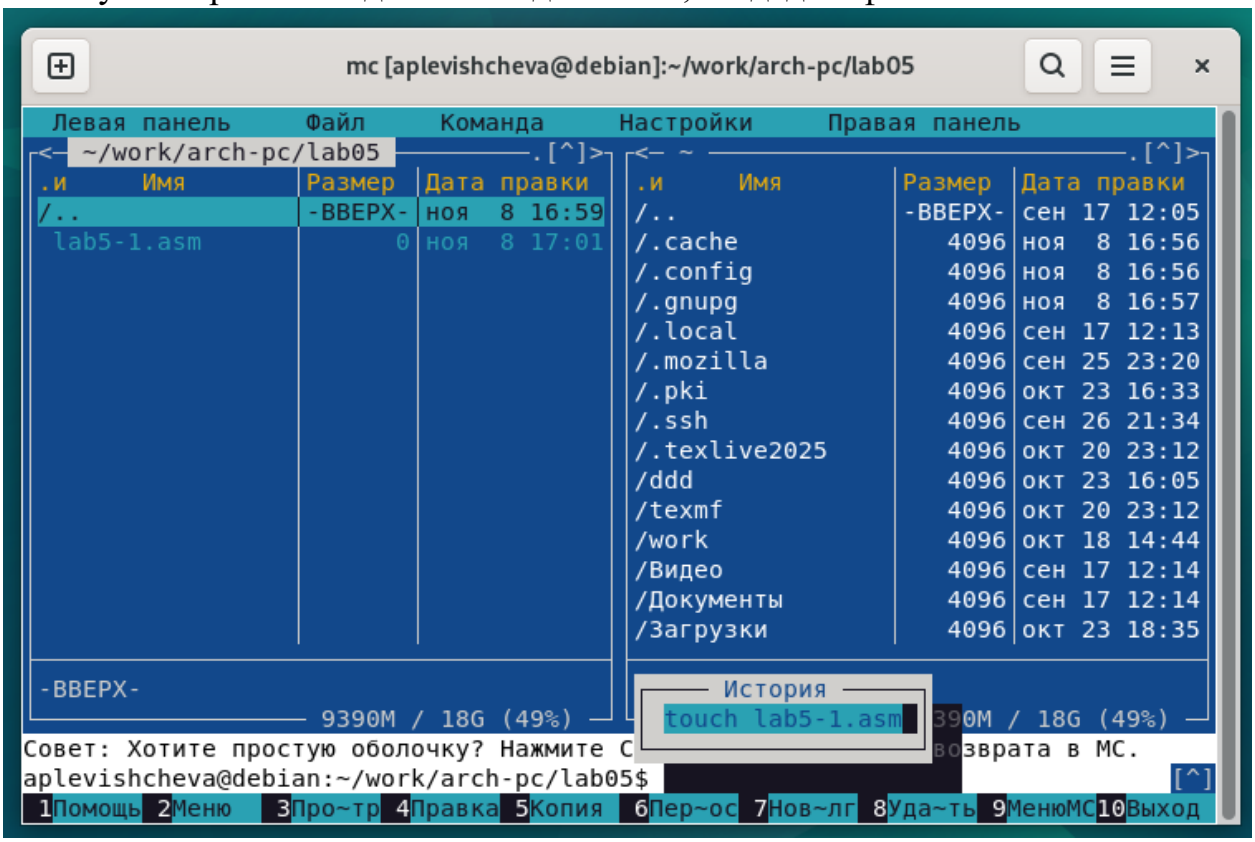

Рис.4.Файл lab5-1.asm

5. С помощью функциональной клавиши F4 откроем файл lab5-1.asm для редактирования во встроенном редакторе mcedit:

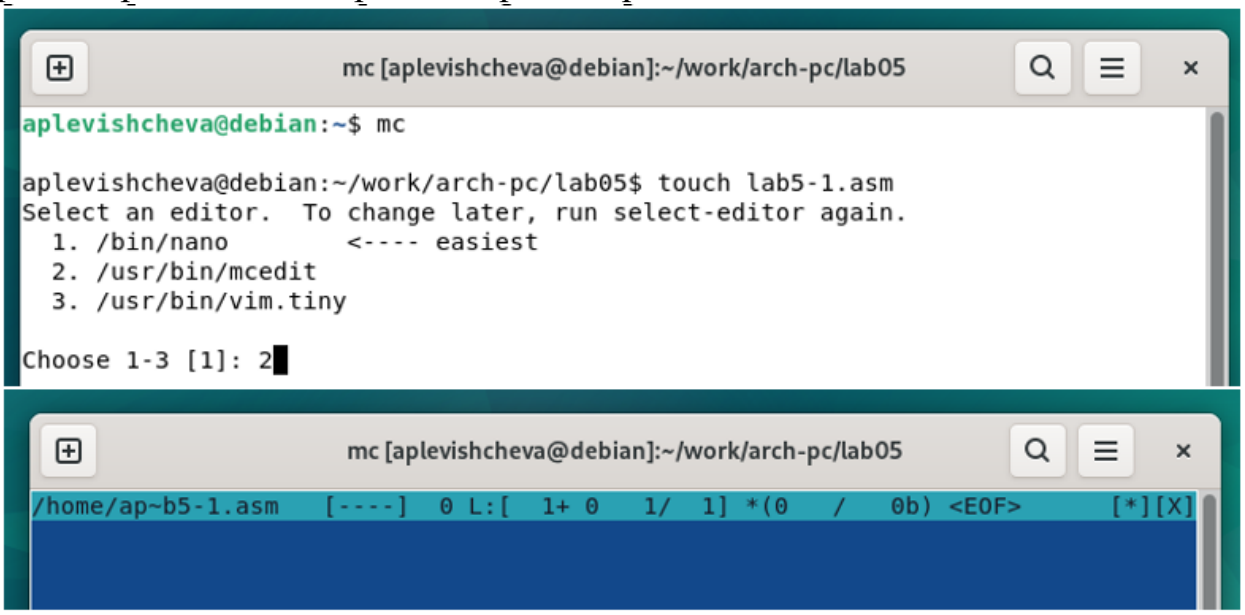

Рис.5.Редактирование

6. Введем текст программы из листинга 5.1:

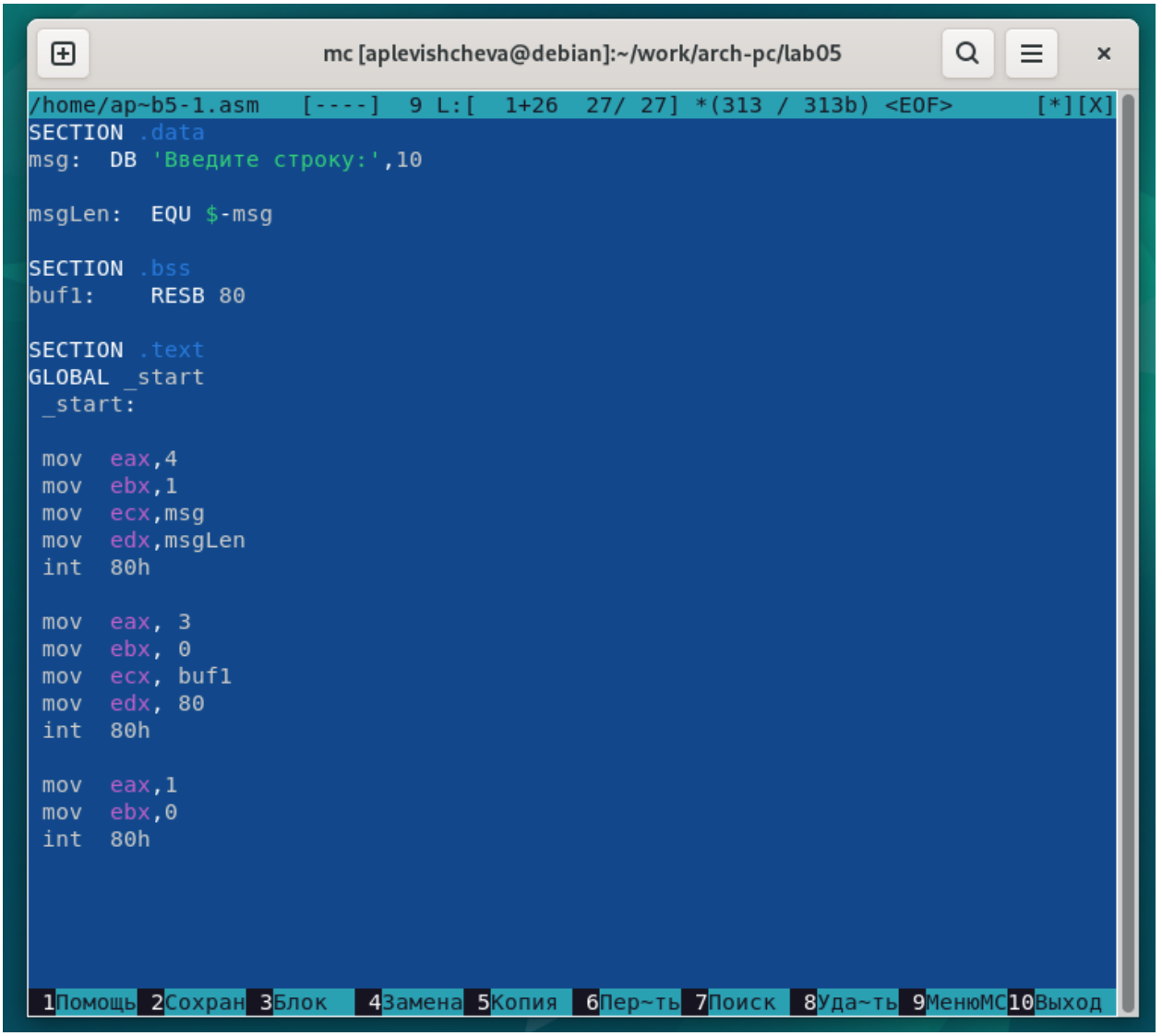

Рис.6.Текст программы

Сохраним изменения и закроем файл.

7. С помощью функциональной клавиши F3 откроем файл lab5-1.asm для просмотра. Убедимся, что файл содержит текст программы:

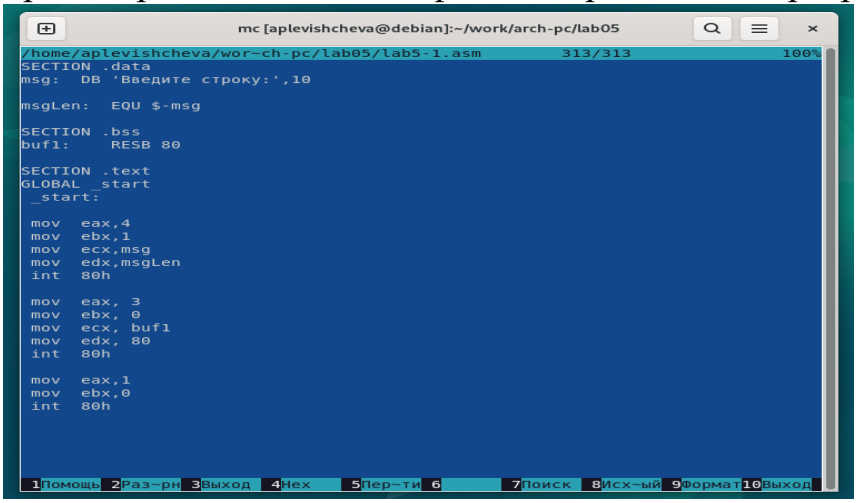

Рис.7.Проверка

8. Оттранслируем текст программы lab5-1.asm в объектный файл, выполним компоновку объектного файла и запустим получившийся исполняемый файл:

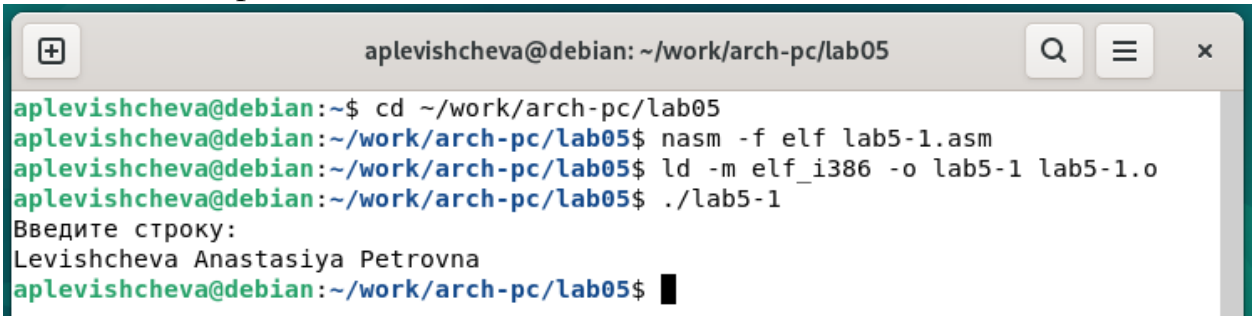

Рис.8.Запуск файла lab5-1

Программа выводит строку 'Введите строку:' и ожидает ввода с клавиатуры. На запрос введено мое ФИО.

9. Скачаем файл in_out.asm со страницы курса в ТУИС:

Рис.9.Загрузка файла

10. Подключаемый файл in_out.asm должен лежать в том же каталоге, что и файл с программой, в которой он используется.

В одной из панелей mc откроем каталог с файлом lab5-1.asm, а в другой панели каталог со скаченным файлом in_out.asm:

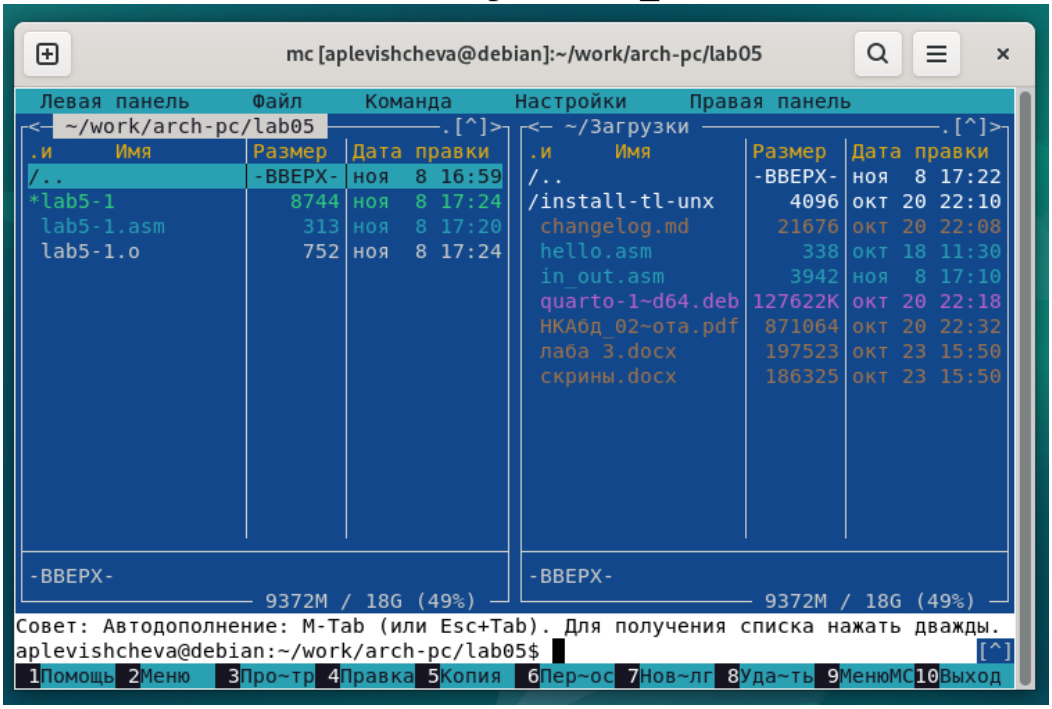

Рис.10.1.Панель mc

Скопируем файл in_out.asm в каталог с файлом lab5-1.asm с помощью функциональной клавиши F5:

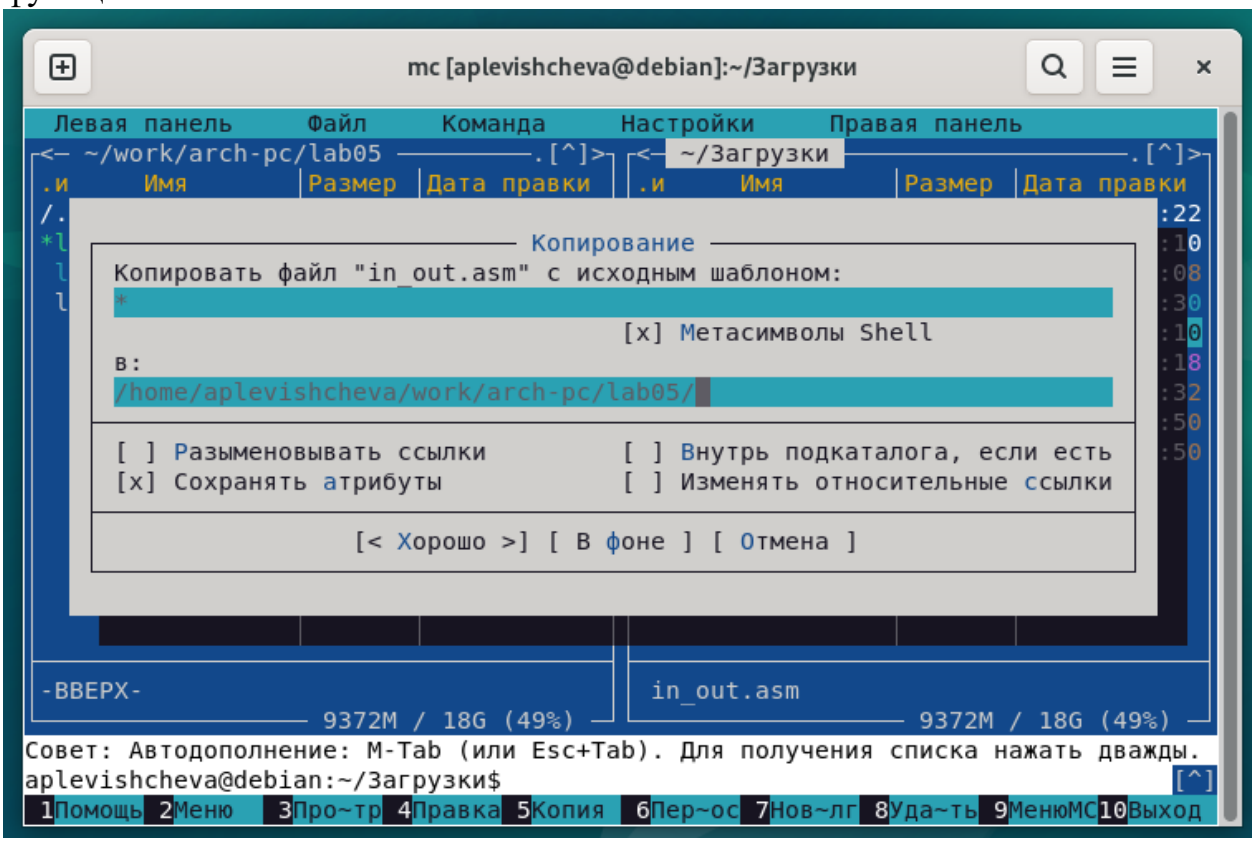

Рис.10.2.Копируем файл

11. С помощью функциональной клавиши F6 создадим копию файла lab5-1.asm с именем lab5-2.asm:

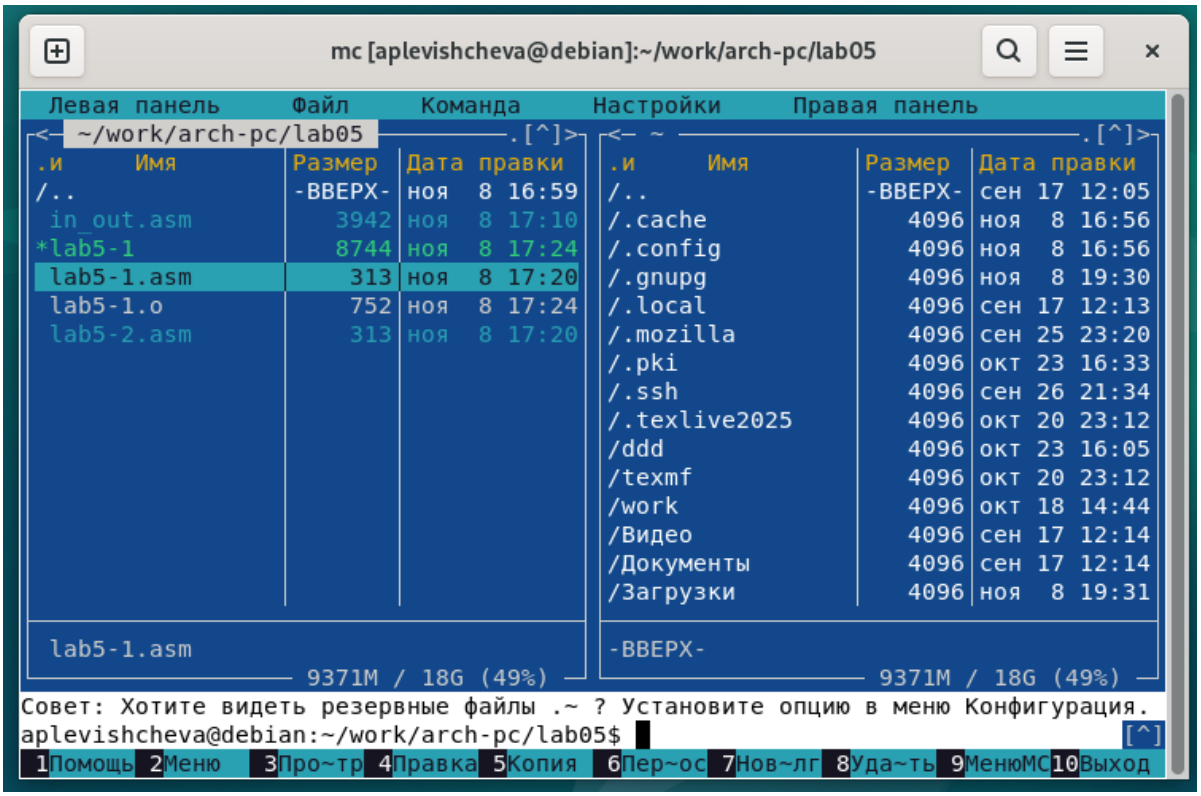

Рис.11.Копия lab5-1.asm

12. Исправим текст программы в файле lab5-2.asm с использованием подпрограмм из внешнего файла in_out.asm (sprintLF, sread и quit) в соответствии с листингом 5.2.

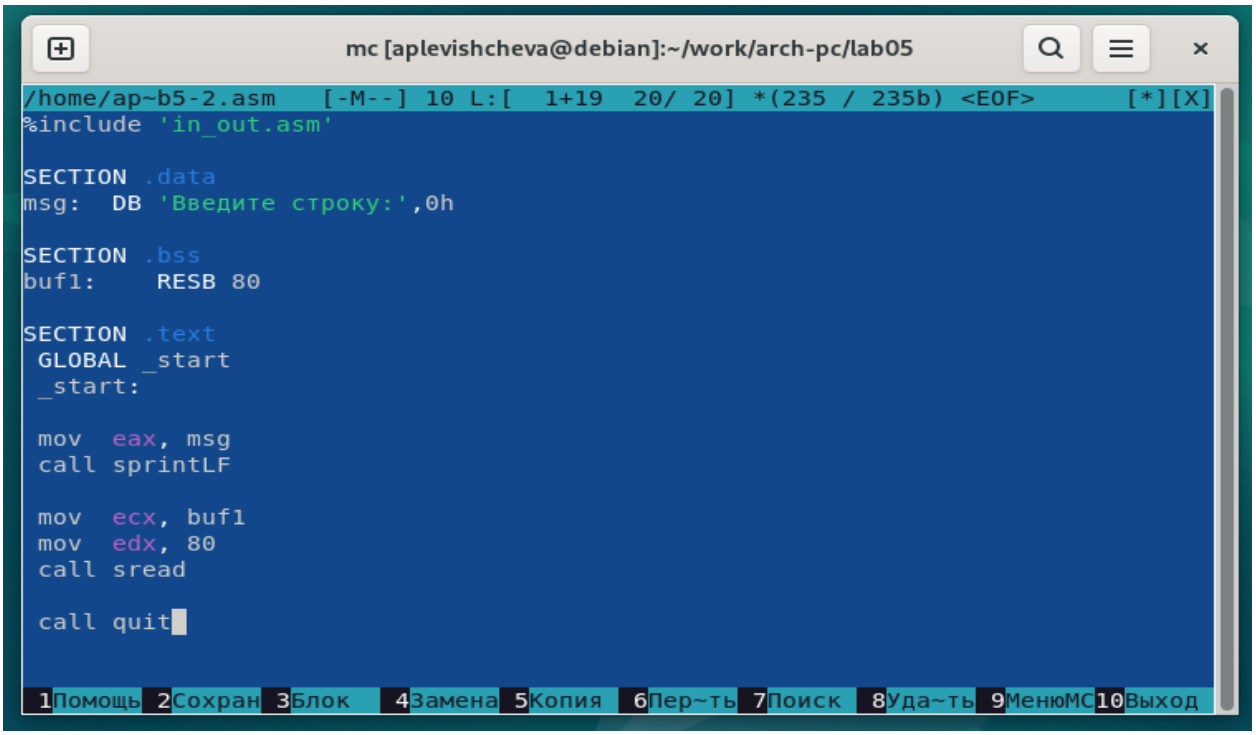

Рис.12.1.Редактирование lab5-2.asm

Создадим исполняемый файл и проверим его работу:

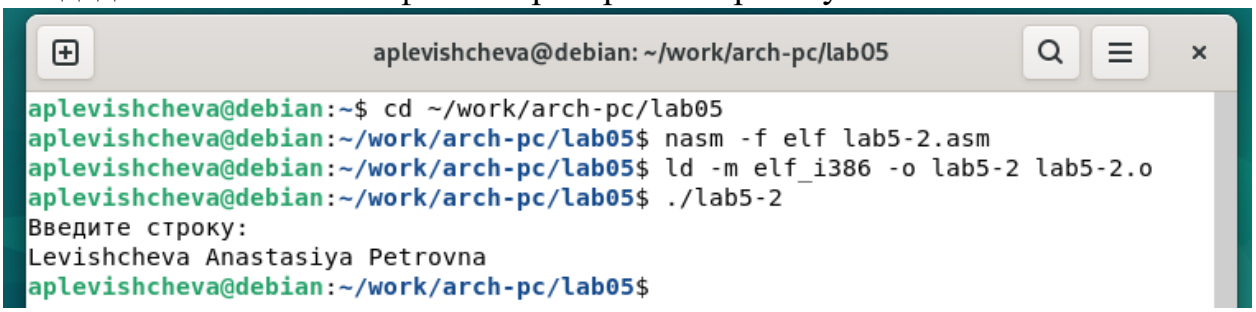

Рис.12.2.Проверка lab5-2.asm

13. В файле lab5-2.asm заменим подпрограмму sprintLF на sprint. Создадим исполняемый файл и проверьте его работу:

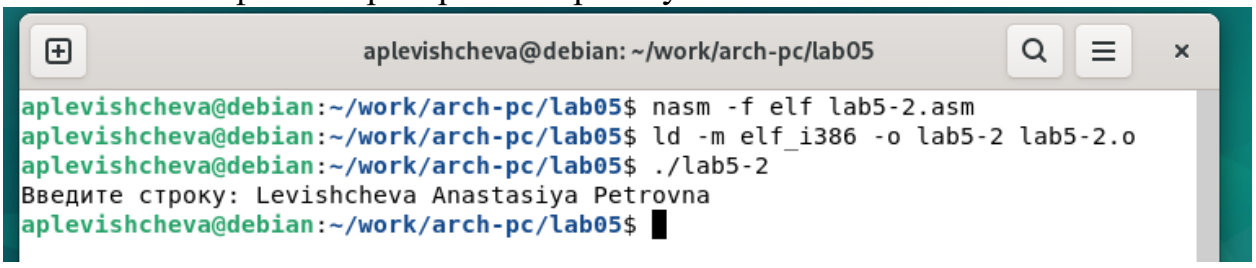

Рис.13.Замена sprintLF на sprint

Разница заключается в том, что при использовании sprint вместо sprintLF ввод строки остается на той же строке что и вывод ‘Введите строку:’.

Задания для самостоятельной работы.

1. Создайте копию файла lab5-1.asm. Внесите изменения в программу (без использования внешнего файла in_out.asm), так чтобы она работала по следующему алгоритму:
   • вывести приглашение типа “Введите строку:”;
   • ввести строку с клавиатуры;
   • вывести введённую строку на экран.

   Создадим копию файла lab5-1.asm и переименуем в lab5-3.asm:

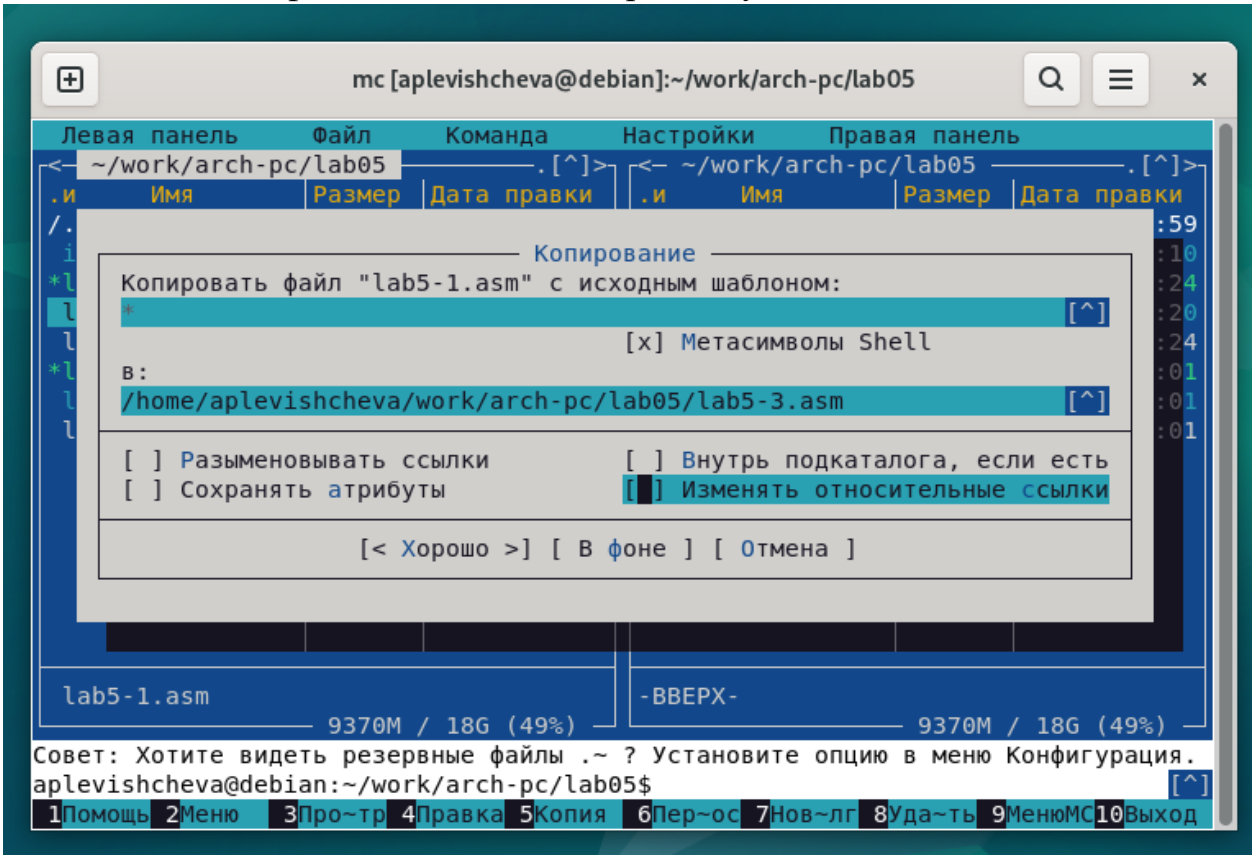

Рис.14.1.Файл lab5-3.asm

Отредактируем исполняемый код, чтобы он соответствовал условию:

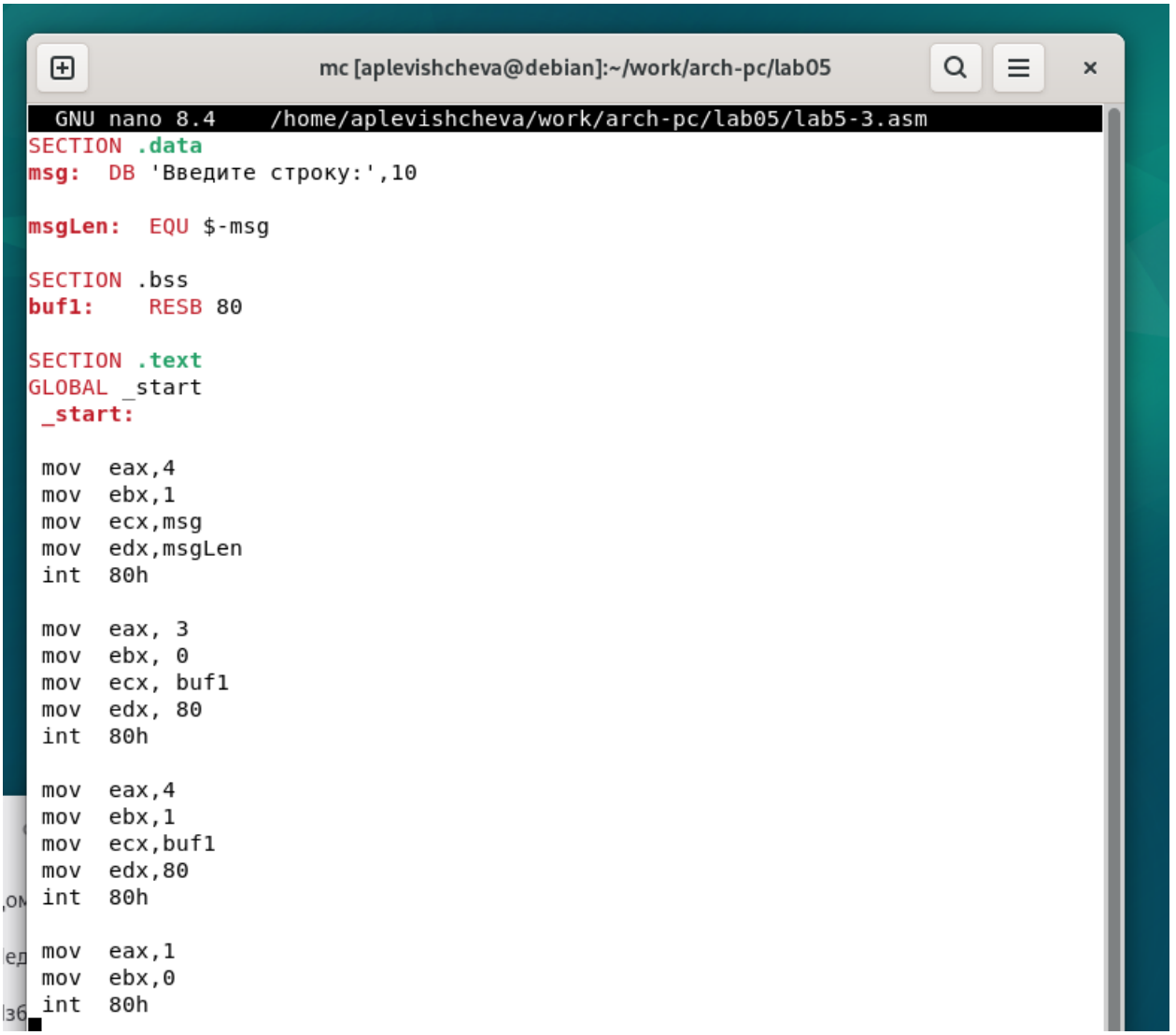

Рис.14.2.Редактируем lab5-3.asm

Проверим работу файла:

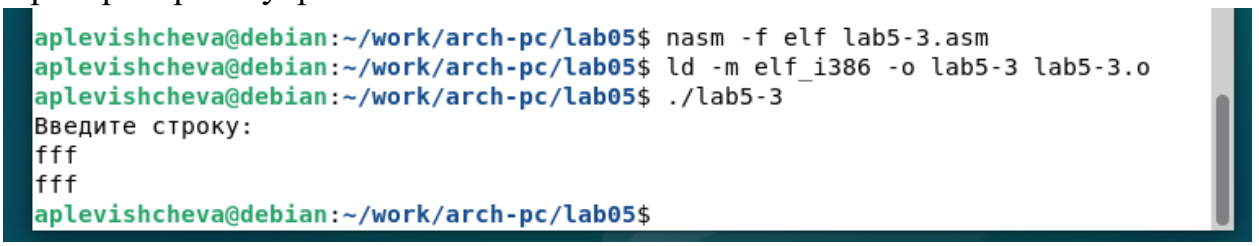

Рис.14.3.Проверка lab5-3.asm

2. Получите исполняемый файл и проверьте его работу. На приглашение ввести строку введите свою фамилию.

   Исполняемый файл был получен во время выполнения задания 1. Проверим работу файла введя мою фамилию:

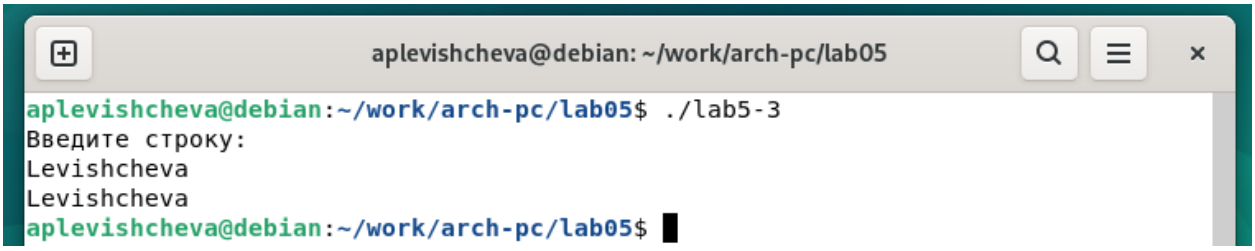

Рис.15.Проверка lab5-3.asm(1)

3. Создайте копию файла lab5-2.asm. Исправьте текст программы с использование подпрограмм из внешнего файла in_out.asm, так чтобы она работала по следующему алгоритму:
• вывести приглашение типа “Введите строку:”;
• ввести строку с клавиатуры;
• вывести введённую строку на экран.

Создадим копию файла lab5-2.asm и переименуем в lab5-4.asm:

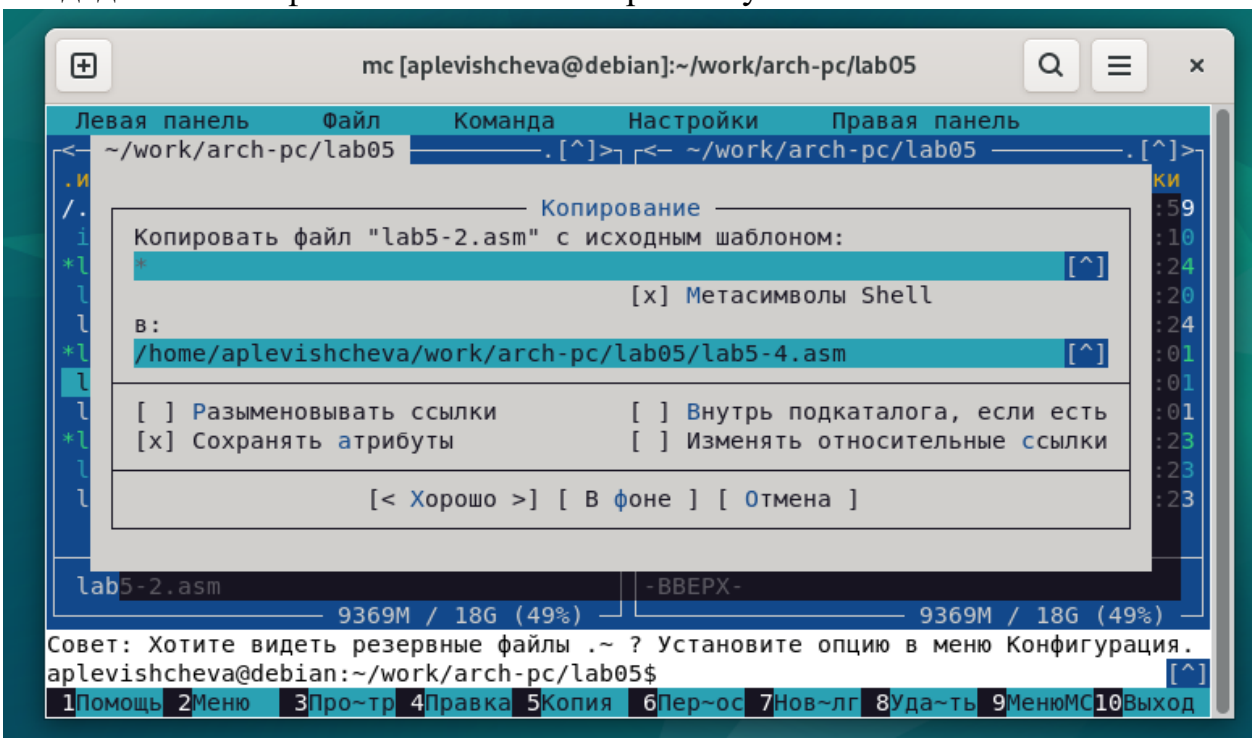

Рис.16.1.Файл lab5-4.asm

Отредактируем исполняемый код, чтобы он соответствовал условию:

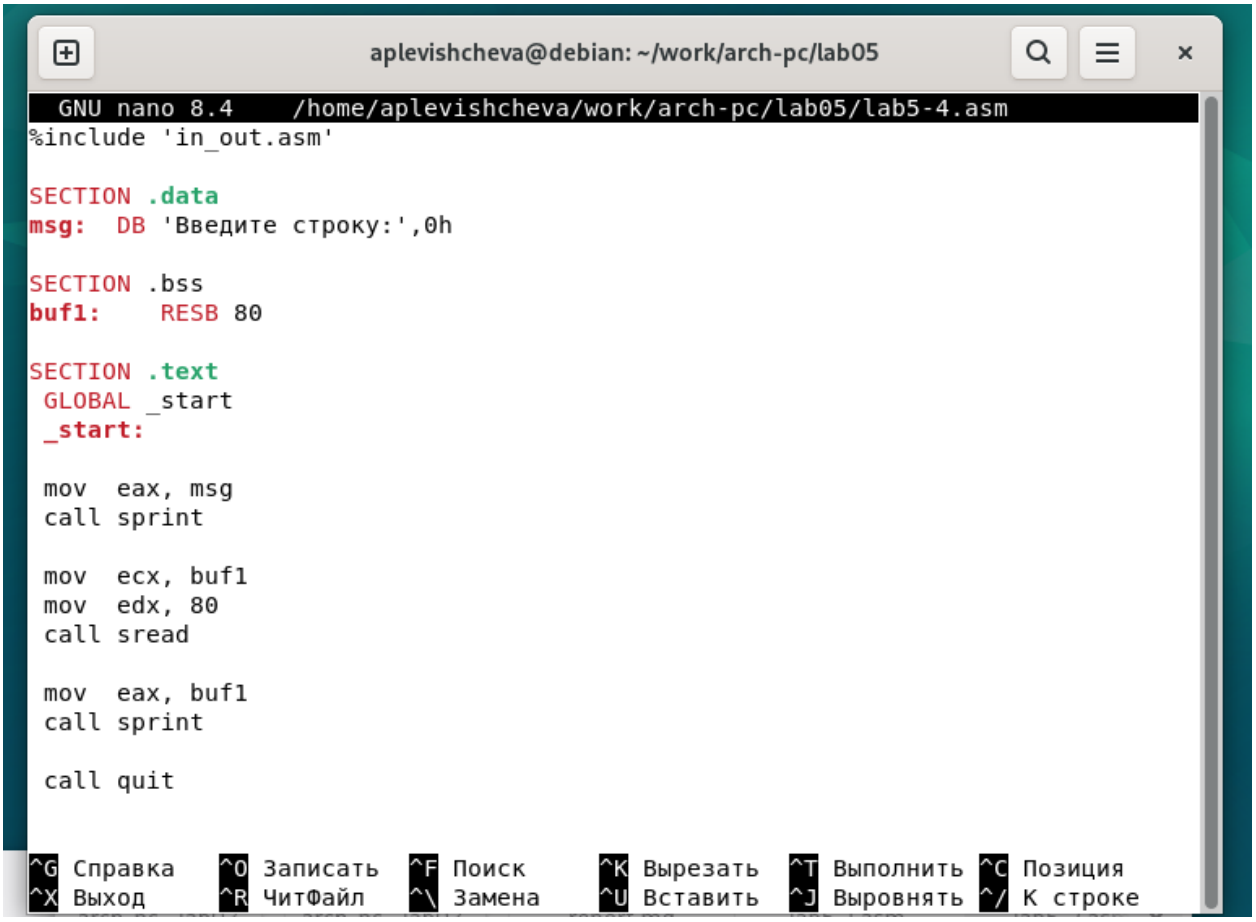

Рис.16.2. Редактируем lab5-4.asm

Проверим работу файла:

```
aplevishcheva@debian:~/work/arch-pc/lab05$ nasm -f elf lab5-4.asm
aplevishcheva@debian:~/work/arch-pc/lab05$ ld -m elf_i386 -o lab5-4 lab5-4.o
aplevishcheva@debian:~/work/arch-pc/lab05$ ./lab5-4
Введите строку:fff
fff
aplevishcheva@debian:~/work/arch-pc/lab05$
```

Рис.16.3.Проверка lab5-4.asm

4. Создайте исполняемый файл и проверьте его работу.

Исполняемый файл был получен во время выполнения задания 3.
Проверим работу файла введя мою фамилию:

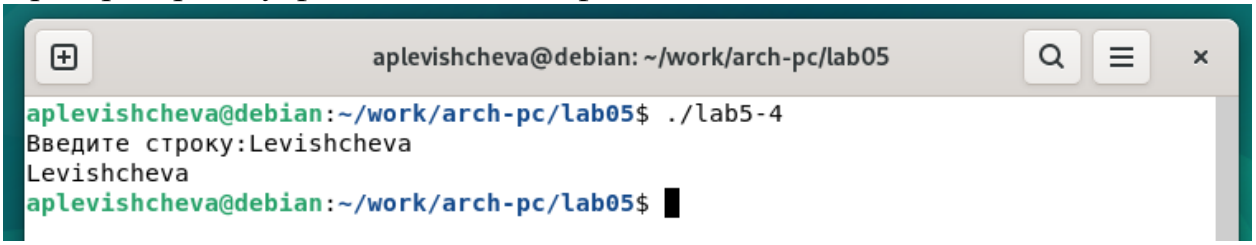

Рис.17. Проверка lab5-4.asm(1)

Выводы

В ходе выполнения лабораторной работы были успешно освоены основы работы с Midnight Commander (mc). Изучена и применена на практике структура программы на языке ассемблера NASM. Также были получены базовые знания о системных вызовах в ОС GNU Linux.

Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.

2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.

3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.

4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.

6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.

7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.

8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.

9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.

10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.

11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.

12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.

13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.

14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL:

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).