

Отчёт по Лабораторной работе №5.

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

Дагделен Зейнап Реджеповна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
3.1	Основы работы с Midnight Commander	7
3.2	Элементы программирования	8
3.2.1	Описание инструкции mov	8
3.2.2	Описание инструкции int	8
3.2.3	Системные вызовы для обеспечения диалога с пользователем	9
4	Выполнение лабораторной работы	10
4.1	Основы работы с tc	10
4.2	Структура программы на языке ассемблера NASM	13
4.3	Подключение внешнего файла	14
4.4	Задание для самостоятельной работы	17
5	Выводы	24
6	Список литературы	25

Список иллюстраций

4.1	Установка команды mc	10
4.2	Окно Midnight Commander.	11
4.3	Окно Midnight Commander. Смена текущего каталога	11
4.4	Окно Midnight Commander. Создание каталога	12
4.5	Окно Midnight Commander. Создание файла	12
4.6	Проверка работы команды	12
4.7	Окно Midnight Commander. Редактор nano	13
4.8	Текст программы в редакторе nano	13
4.9	Транслирование текста программы в объектный файл	14
4.10	Компоновка объектного файла	14
4.11	Запуск программы	14
4.12	Файл in_out.asm	15
4.13	Копирование файла из одного каталога в другой	15
4.14	Окно Midnight Commander. Создание копии файла	16
4.15	Транслирование текста программы в объектный файл	16
4.16	Компоновка объектного файла	16
4.17	Работа программы	17
4.18	Транслирование текста программы в объектный файл	17
4.19	Компоновка объектного файла	17
4.20	Работа программы	17
4.21	Копирование файла под другим именем	18
4.22	Текст программы	19
4.23	Транслирование текста программы в объектный файл	20
4.24	Компоновка объектного файла	20
4.25	Работа программы	20
4.26	Создание копии файла под другим именем	21
4.27	Текст программы	22
4.28	Транслирование текста программы в объектный файл	23
4.29	Компоновка объектного файла	23
4.30	Работа программы	23

Список таблиц

1 Цель работы



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

2 Задание

1. Основы работы с тс
2. Структура программы на языке ассемблера NASM
3. Подключение внешнего файла
4. Выполнение заданий для самостоятельной работы

3 Теоретическое введение

3.1 Основы работы с Midnight Commander

Midnight Commander — 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. Некоторые комбинации клавиш облегчают работу с Midnight Commander, например: - Tab используется для переключения между панелями; -  и  используется для навигации, Enter для входа в каталог или открытия файла; • Ctrl + u меняет местами содержимое правой и левой панелей; и другие ## 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).

Для объявления инициированных данных в секции .data используются директивы: - 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.

3.2 Элементы программирования

3.2.1 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).

3.2.2 Описание инструкции int

Инструкция языка ассемблера `int` предназначена 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).

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`.

3.2.3 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.

4 Выполнение лабораторной работы

4.1 Основы работы с mc

1. Открываю Midnight Commander с помощью mc (рис. [4.2]). Но так как эта команда у меня не установлена, следуя инструкциям и скачиваю его(рис. [4.1])

```
zrdagdelen@zrdagdelen:~$ mc
Команда «mc» не найдена, но может быть установлена с помощью:
sudo apt install mc
zrdagdelen@zrdagdelen:~$ sudo apt install mc
[sudo] пароль для zrdagdelen: 
```

Рис. 4.1: Установка команды mc

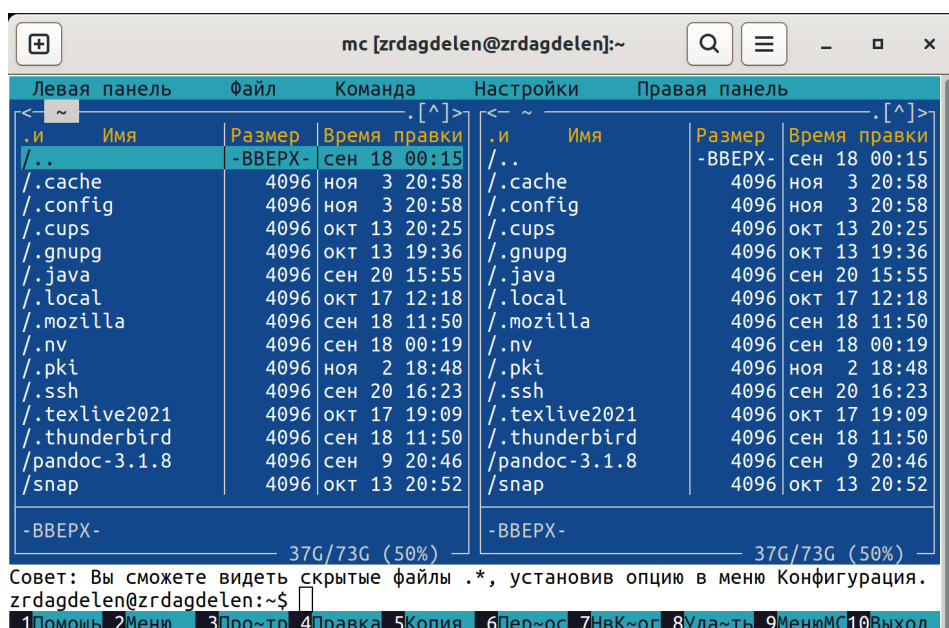


Рис. 4.2: Окно Midnight Commander.

2. Пользуясь клавишами **⌘**, **⌘** и Enter, перехожу в каталог ~/work/arch-pc созданный при выполнении лабораторной работы №4 (рис. [4.3]).

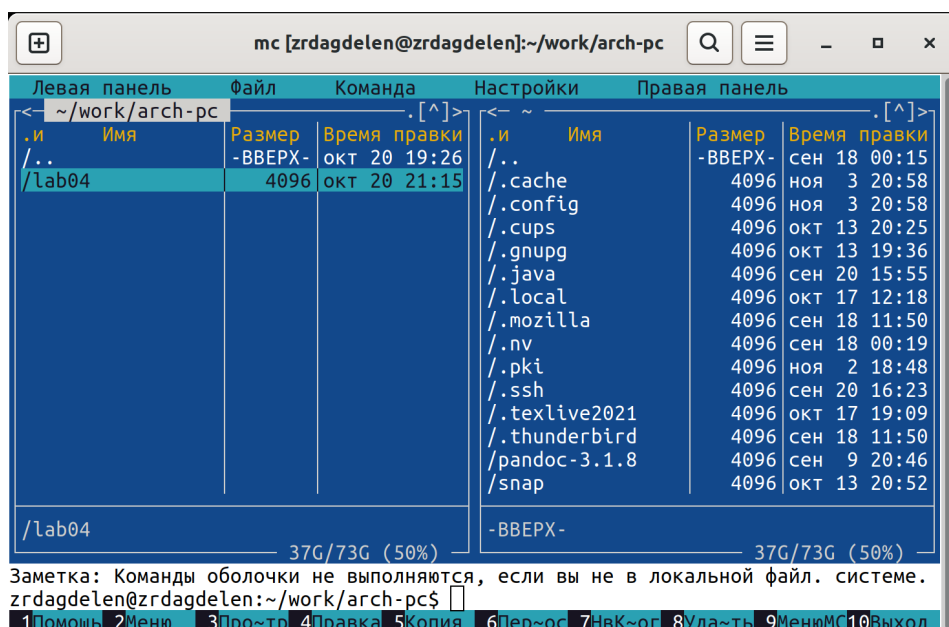


Рис. 4.3: Окно Midnight Commander. Смена текущего каталога

3. С помощью функциональной клавиши F7 создаю папку lab05 (рис. [4.4]) и перехожу в созданный каталог.

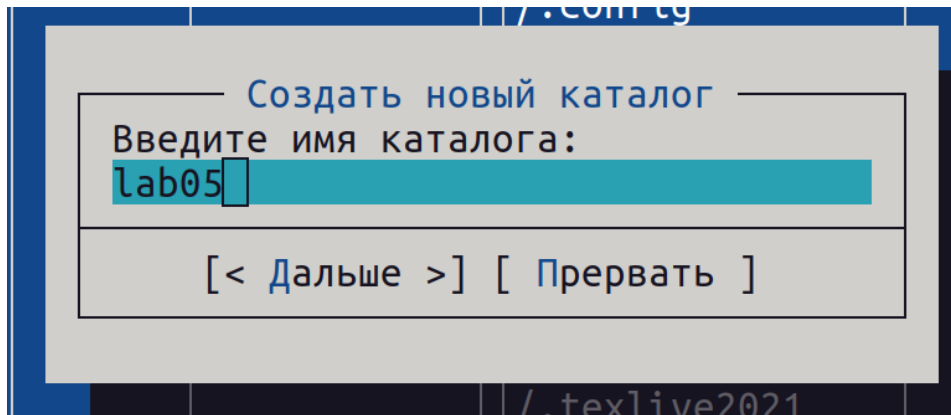


Рис. 4.4: Окно Midnight Commander. Создание каталога

4. Пользуясь строкой ввода и командой touch создаю файл lab5-1.asm (рис. [4.5]). Проверяем (рис. [4.6])

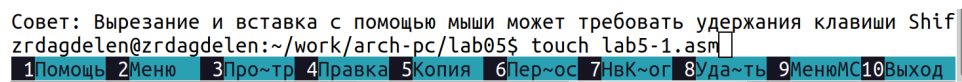


Рис. 4.5: Окно Midnight Commander. Создание файла

Левая панель		Файл	Команда	
< ~/work/arch-pc/lab05			.[^]>	
.и	Имя	Размер	Время правки	
/..		-ВВЕРХ-	ноя 3	21:02
lab5-1.asm		0	ноя 3	21:11

Рис. 4.6: Проверка работы команды

4.2 Структура программы на языке ассемблера NASM

5. С помощью функциональной клавиши F4 открываю файл lab5-1.asm для редактирования во встроенном редакторе. В качестве встроенного редактора Midnight Commander использую редактор nano (рис. [4.7])

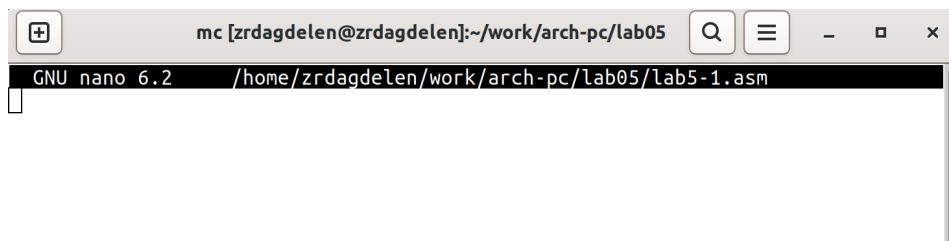


Рис. 4.7: Окно Midnight Commander. Редактор nano

6. Ввожу текст программы из листинга 5.1, данного в PDF-файле по лабораторной №5 в курсе Архитектура ЭВМ, сохраняю изменения и закрываю файл (для редактора nano: Ctrl + x (выход) > Y (сохранить изменения) > Enter) (рис. [4.8]).

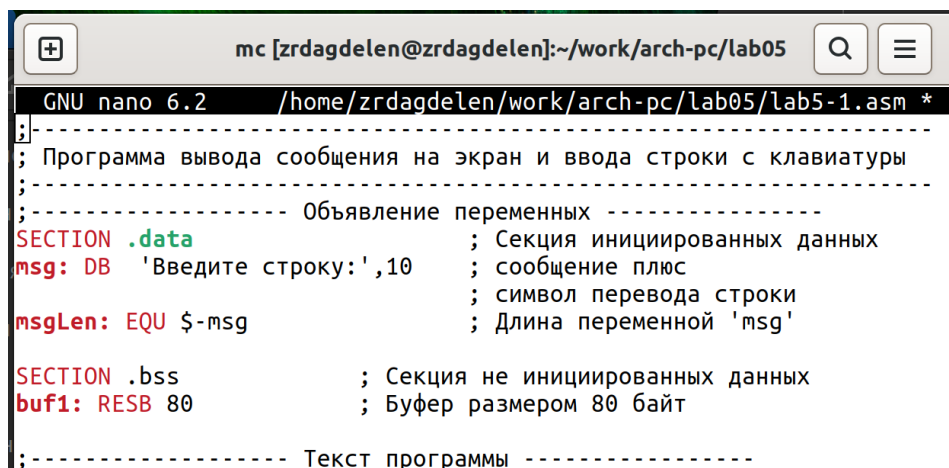
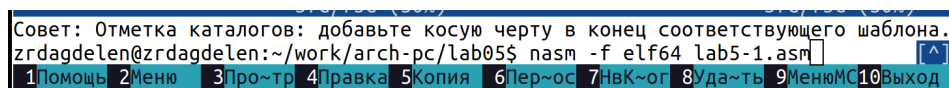


Рис. 4.8: Текст программы в редакторе nano

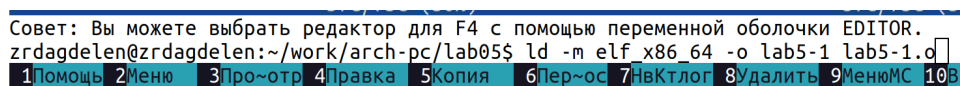
7. Оттранслирую текст программы lab5-1.asm в объектный файл с помощью команды `nasm -f` (так как у меня 64-битная версия, использую `elf64`)

(рис.[4.9]). Выполняю компоновку объектного файла с помощью `ld -m` (так как у меня архитектура Линукса x86, использую `elf_x86_64`) (рис. [4.10])и запускаю получившийся исполняемый файл . Программа выводит строку ‘Введите строку:’ и ожидает ввода с клавиатуры. На этот запрос ввожу свое ФИО: Дагделен Зейнап Реджеповна (рис.[4.11]).



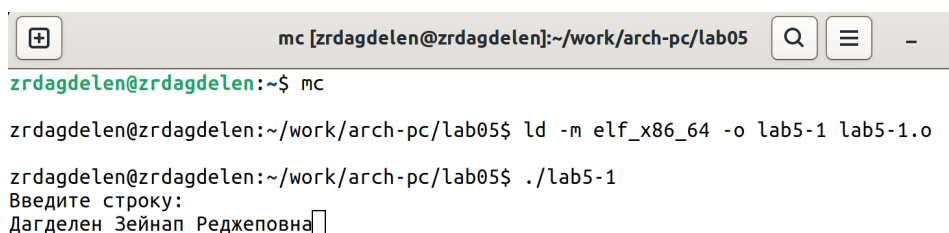
```
Совет: Отметка каталогов: добавьте косую черту в конец соответствующего шаблона.
zrdagdelen@zrdagdelen:~/work/arch-pc/lab05$ nasm -f elf64 lab5-1.asm
```

Рис. 4.9: Транслирование текста программы в объектный файл



```
Совет: Вы можете выбрать редактор для F4 с помощью переменной оболочки EDITOR.
zrdagdelen@zrdagdelen:~/work/arch-pc/lab05$ ld -m elf_x86_64 -o lab5-1 lab5-1.o
```

Рис. 4.10: Компоновка объектного файла



```
mc [zrdagdelen@zrdagdelen]:~/work/arch-pc/lab05
zrdagdelen@zrdagdelen:~$ mc
zrdagdelen@zrdagdelen:~/work/arch-pc/lab05$ ld -m elf_x86_64 -o lab5-1 lab5-1.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab05$ ./lab5-1
Введите строку:
Дагделен Зейнап Реджеповна
```

Рис. 4.11: Запуск программы

4.3 Подключение внешнего файла

8. Скачиваю файл `in_out.asm` со страницы курса в ТУИС. Теперь этот файл в папке “Загрузки” (рис.[4.12]).

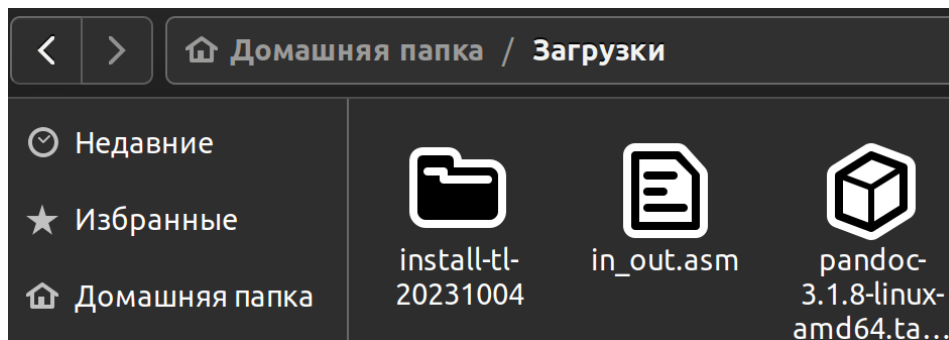


Рис. 4.12: Файл in_out.asm

9. Копирую с помощью функциональной клавиши F5 подключаемый файл in_out.asm в тот же каталог, что и файл с программой, в которой он используется (рис. [4.13]).

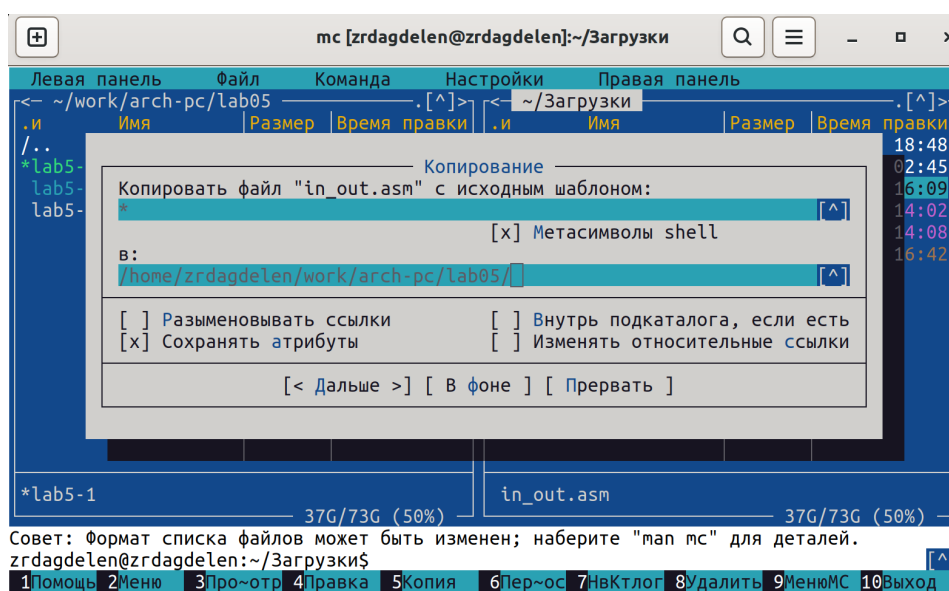


Рис. 4.13: Копирование файла из одного каталога в другой

10. С помощью функциональной клавиши F6 создаю копию файла lab5-1.asm с именем lab5-2.asm. Выделяю файл lab5-1.asm, нажимаю клавишу F6, ввожу имя файла lab5-2.asm и нажимаю клавишу Enter (рис. [4.14]).

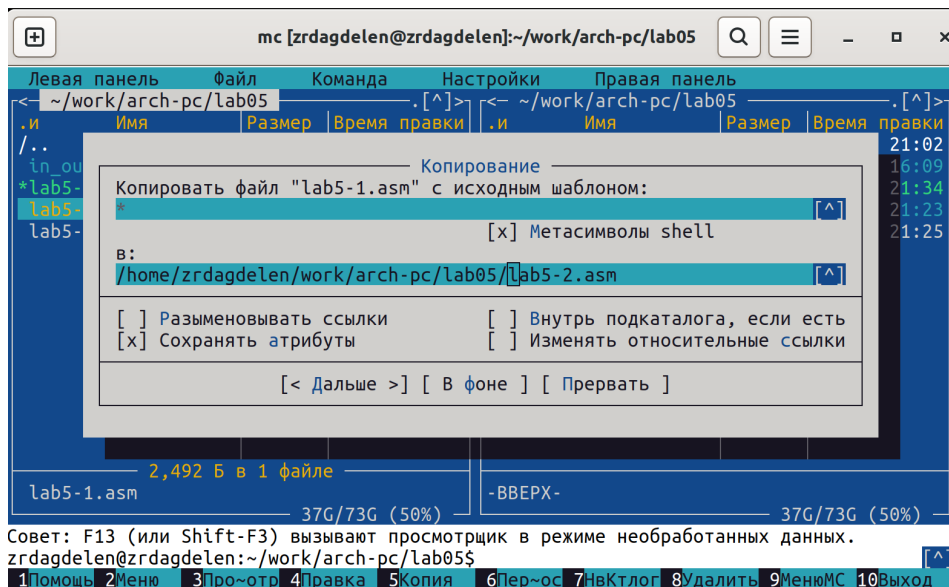


Рис. 4.14: Окно Midnight Commander. Создание копии файла

11. Исправляю текст программы в файле lab5-2.asm с использованием подпрограмм из внешнего файла in_out.asm (использую подпрограммы sprintLF, sread и quit) в соответствии с листингом 5.2. (рис. [??]). Создаю исполняемый файл (рис.[4.15] - рис.[4.16]) и проверяю его работу (рис. [4.17]).

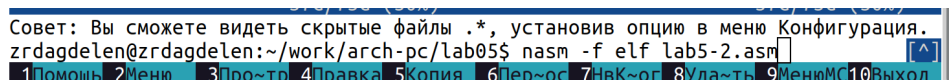


Рис. 4.15: Транслирование текста программы в объектный файл

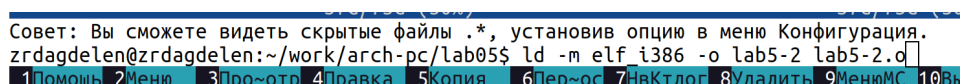


Рис. 4.16: Компоновка объектного файла


```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab05$
Введите строку:
Дагделен Зейнап Реджеповна
```

Рис. 4.17: Работа программы

12. В файле lab5-2.asm заменяю подпрограмму sprintLF на sprint. Создаю исполняемый файл (рис.[4.18] - рис.[4.19]) и проверяю его работу (рис. [4.20]). В чем разница? Разница в том, что в первом случае (с подпрограммой sprintLF): строка вводится на следующей строке (то есть программа принимает мое ФИО с новой строки); во втором случае: все вводится и выводится на одной строке.

```
Совет: Вы сможете видеть скрытые файлы .*, установив опцию в меню Конфигурация.
zrdagdelen@zrdagdelen:~/work/arch-pc/lab05$ nasm -f elf lab5-2.asm
```

Рис. 4.18: Транслирование текста программы в объектный файл

```
Совет: Вы сможете видеть скрытые файлы .*, установив опцию в меню Конфигурация.
zrdagdelen@zrdagdelen:~/work/arch-pc/lab05$ ld -m elf_i386 -o lab5-2 lab5-2.o
```

Рис. 4.19: Компоновка объектного файла

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab05$ ./lab5-2
Введите строку: Дагделен Зейнап Реджеповна
```

Рис. 4.20: Работа программы

4.4 Задание для самостоятельной работы

1. Создаю копию файла lab5-1.asm, для удобства переименовываю (рис.[4.21]). Вношу изменения в программу (без использования внешнего файла

in_out.asm), так чтобы она работала по следующему алгоритму (рис.[4.22]):

- выводит приглашение типа “Введите строку:”;
- вводит строку с клавиатуры;
- выводит введенную строку на экран.

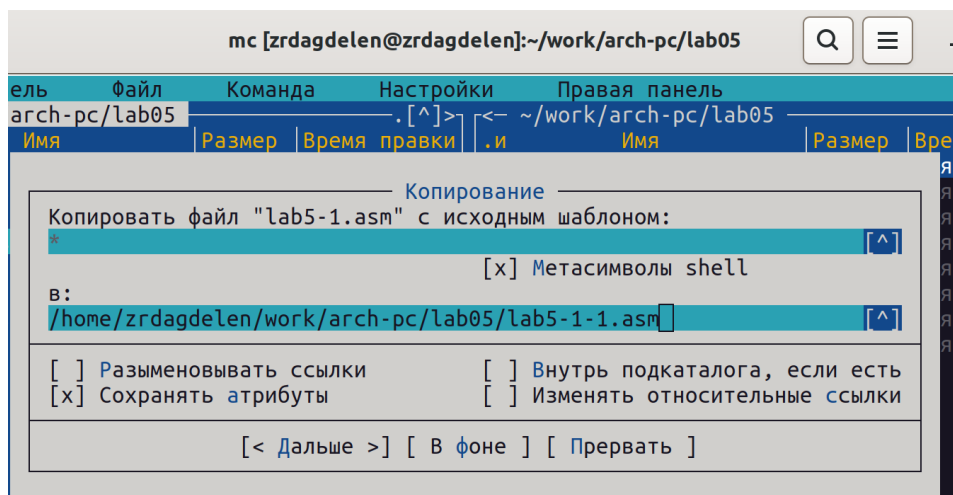
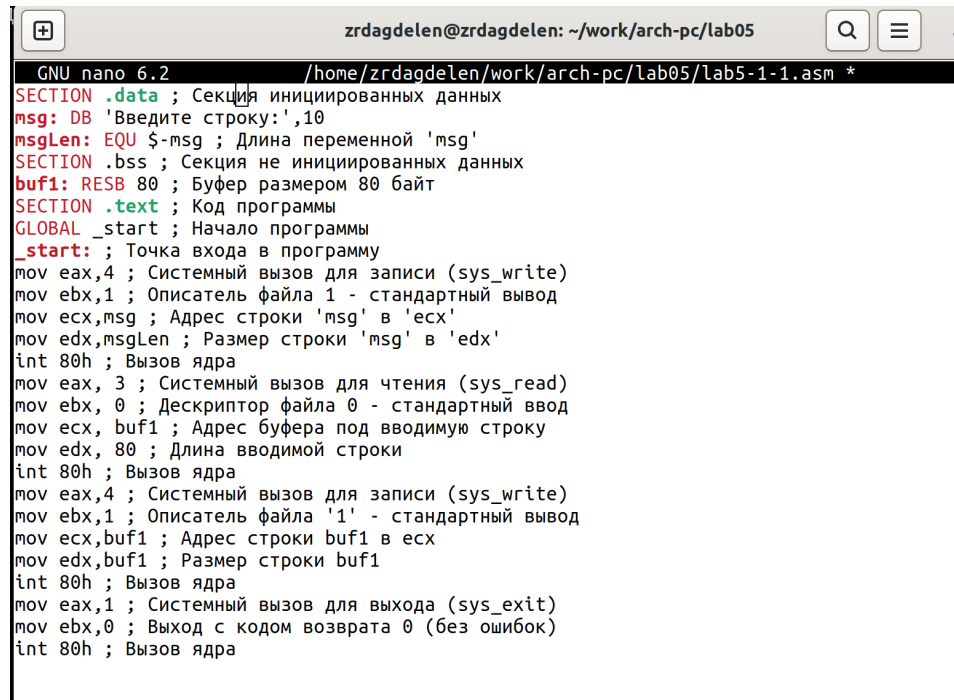


Рис. 4.21: Копирование файла под другим именем



```
GNU nano 6.2 /home/zrdagdelen/work/arch-pc/lab05/Lab5-1-1.asm *
SECTION .data ; Секция инициализированных данных
msg: DB 'Введите строку:',10
msgLen: EQU $-msg ; Длина переменной 'msg'
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,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
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
mov eax,4 ; Системный вызов для записи (sys_write)
mov ebx,1 ; Описатель файла '1' - стандартный вывод
mov ecx,buf1 ; Адрес строки buf1 в ecx
mov edx,buf1 ; Размер строки buf1
int 80h ; Вызов ядра
mov eax,1 ; Системный вызов для выхода (sys_exit)
mov ebx,0 ; Выход с кодом возврата 0 (без ошибок)
int 80h ; Вызов ядра
```

Рис. 4.22: Текст программы

Текст программы:

```
SECTION .data ; Секция инициализированных данных
msg: DB 'Введите строку:',10
msgLen: EQU $-msg ; Длина переменной 'msg'
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,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
```

```

mov eax, 3 ; Системный вызов для чтения (sys_read)
mov ebx, 0 ;Descriptor файла 0 - стандартный ввод
mov ecx, buf1 ; Адрес буфера под вводимую строку
mov edx, 80 ; Длина вводимой строки
int 80h ; Вызов ядра
mov eax,4 ; Системный вызов для записи (sys_write)
mov ebx,1 ; Описатель файла '1' - стандартный вывод
mov ecx,buf1 ; Адрес строки buf1 в ecx
mov edx,buf1 ; Размер строки buf1
int 80h ; Вызов ядра
mov eax,1 ; Системный вызов для выхода (sys_exit)
mov ebx,0 ; Выход с кодом возврата 0 (без ошибок)
int 80h ; Вызов ядра

```

2. Получаю исполняемый файл (рис.[4.23]-рис.[4.24]) и проверяю его работу (рис.[4.25]).

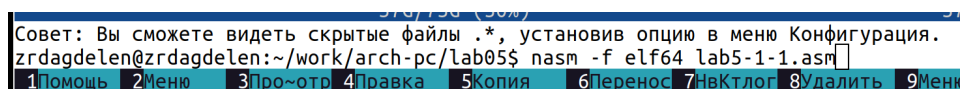


Рис. 4.23: Транслирование текста программы в объектный файл

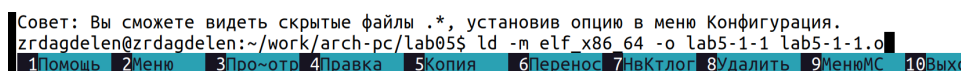


Рис. 4.24: Компоновка объектного файла

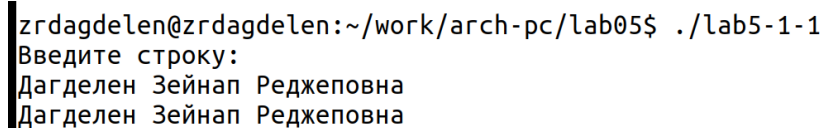


Рис. 4.25: Работа программы

3. Создаю копию файла lab5-2.asm с именем lab5-2-1 (рис. [4.26]). Исправляю текст программы с использованием подпрограмм из внешнего файла in_out.asm, так чтобы она работала такому же алгоритму, как и программа lab5-1-1 (рис.[4.27]).

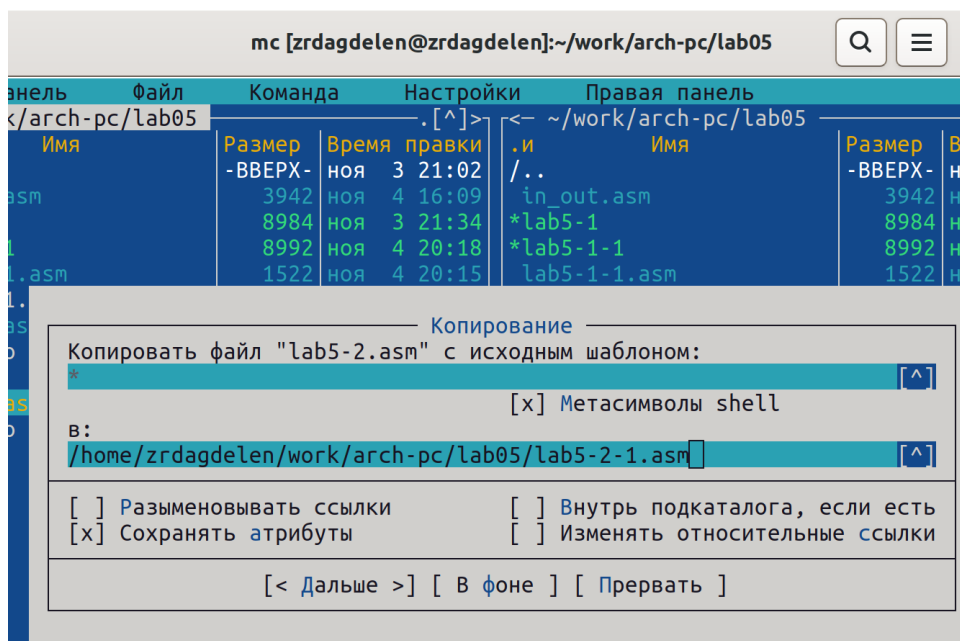


Рис. 4.26: Создание копии файла под другим именем

```
GNU nano 6.2 /home/zrdagdelen/work/arch-pc/Lab6
%include 'in_out.asm'
SECTION .data ; Секция инициированных данных
msg: DB 'Введите строку: ',0h ; сообщение
SECTION .bss ; Секция не инициированных 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
mov eax,4 ; Системный вызов для записи (sys_write)
mov ebx,1 ; Описатель файла '1' - стандартный вывод
mov ecx,buf1 ; Адрес строки buf1 в ecx
int 80h ; Вызов ядра
call quit ; вызов подпрограммы завершения
```

Рис. 4.27: Текст программы

Текст программы:

```
%include 'in_out.asm'

SECTION .data ; Секция инициированных данных
msg: DB 'Введите строку: ',0h ; сообщение

SECTION .bss ; Секция не инициированных 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
```

```

mov eax,4 ; Системный вызов для записи (sys_write)
mov ebx,1 ; Описатель файла '1' - стандартный вывод
mov ecx,buf1 ; Адрес строки buf1 в ecx
int 80h ; Вызов ядра
call quit ; вызов подпрограммы завершения

```

4. Создаю исполняемый файл (рис.[4.28]-рис.[4.29]) и проверяю его работу(рис.[4.30]).

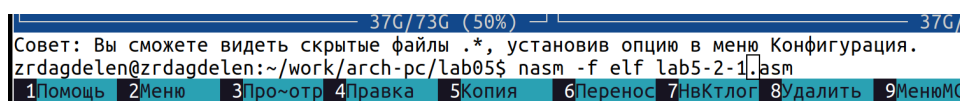


Рис. 4.28: Транслирование текста программы в объектный файл

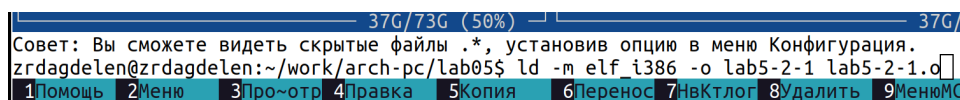


Рис. 4.29: Компоновка объектного файла

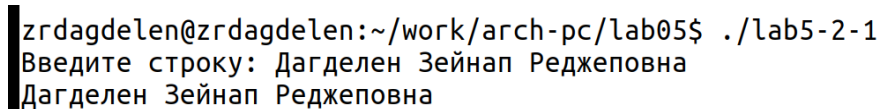


Рис. 4.30: Работа программы

5 Выводы

Я приобрела практические навыки работы в Midnight Commander и освоила инструкции языка ассемблера `mov` и `int`.

6 Список литературы

Архитектура ЭВМ