

Отчёт по лабораторной работе №5

Дисциплина: архитектура компьютера

Жернаков Данила Иванович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Основы работы с тс	9
4.2	Структура программы на языке ассемблера NASM	9
4.3	Выполнение заданий для самостоятельной работы	12
5	Выводы	15

Список иллюстраций

4.1	Открытый Midnight Commander	9
4.2	Копирование файла	10
4.3	Редактирование файла	11
4.4	Исполнение файла	11
4.5	Исполнение файла	11
4.6	Исполнение файла	12
4.7	Редактирование файла	13

Список таблиц

1 Цель работы

Целью данной лабораторной работы является 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

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. Программа на языке ассемблера NASM, как правило, состоит из трёх секций: секция кода программы (SECTION .text), секция иницированных (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 иницированных 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 (четырёх- рённое 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. Инструкция языка ассемблера mov предназначена для дублирования данных источника в приёмнике.

```
mov dst,src
```

Здесь операнд `dst` — приёмник, а `src` — источник. В качестве операнда могут выступать регистры (`register`), ячейки памяти (`memory`) и непосредственные значения (`const`). Инструкция языка ассемблера `int` предназначена для вызова прерывания с указанным номером.

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).

4 Выполнение лабораторной работы

4.1 Основы работы с mc

Открываю Midnight Commander, введя в терминал mc (рис. 4.1).

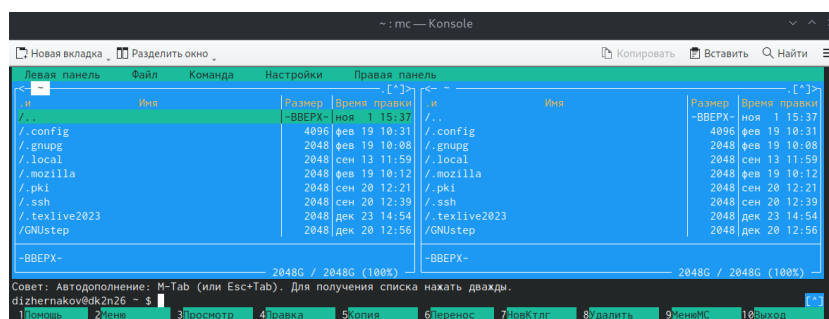
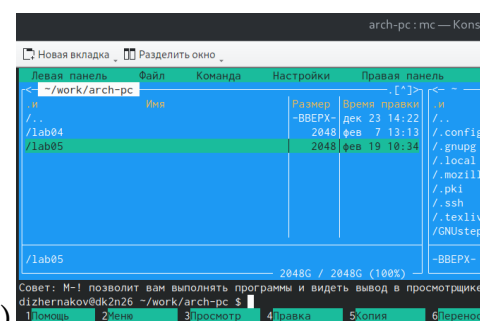


Рис. 4.1: Открытый Midnight Commander



Перешел в каталог, создал папку и создал файл (рис. @fig:002)

4.2 Структура программы на языке ассемблера NASM

С помощью функциональной клавиши F4 открываю созданный файл для редактирования в редакторе mcedit. Ввожу в файл код программы для запроса строки

lab05: mc — Console

Новая вкладка

Разделить окно

Копировать

Вставить

Найти

lab5-1.asm [-M-] 33 LqL 1+10 11/ 32) x(669 /21756) 0010 0x00A

1; ----- Объявление переменных -----

2 SECTION .data ; Секция иницированных данных

3 msg: DB 'Введите строку:',10 ; сообщение плюс

4 ; символ перевода строки

5 msgLen: EQU \$-msg ; Длина переменной 'msg'

6 SECTION .bss ; Секция не иницированных данных

7 buf1: RESB 80 ; Буфер размером 80 байт

8 ; ----- Текст программы -----

9 SECTION .text ; Код программы

10 GLOBAL _start ; Начало программы

11 _start: ; Точка входа в программу

12 ; ----- Системный вызов 'write'

13 ; После вызова инструкции 'int 80h' на экран будет

14 ; выведено сообщение из переменной 'msg' длиной 'msgLen'

15 mov eax,4 ; Системный вызов для записи (sys_write)

16 mov ebx,1 ; Описатель файла 1 - стандартный вывод

1.Понедельник

2.Вторник

3.Среда

4.Четверг

5.Пятница

6.Суббота

7.Воскресенье

8.Понедельник

9.Вторник

10.Выход

```

dizhernakov@dk2n26 ~ % cd work/arch-pc/
dizhernakov@dk2n26 ~/work/arch-pc % cd lab05
dizhernakov@dk2n26 ~/work/arch-pc/lab05 $ ls
lab5-1.asm
dizhernakov@dk2n26 ~/work/arch-pc/lab05 % nasm -f elf lab5-1.asm
dizhernakov@dk2n26 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-1 lab5-1.o
dizhernakov@dk2n26 ~/work/arch-pc/lab05 $ ls
lab5-1 lab5-1.asm lab5-1.o

```

```
dizhernakov@dk2n26 ~/work/arch-pc/lab05 $ ./lab5-1
Введите строку:
Хернаков Д
dizhernakov@dk2n26 ~/work/arch-pc/lab05 $
```

Скачиваю файл in_out.asm со страницы курса в ТУИС, копирую файл in_out.asm из каталога Загрузки в созданный каталог lab05 и копирую файл lab5-1 с другим именем

```
dizhernakov@dk2n26 ~/work/arch-pc/lab9$ cp ~/Загрузки/in_out.asm in_out.asm
dizhernakov@dk2n26 ~/work/arch-pc/lab9$ ls
in_out.asm  lab5-1      lab5-1.asm  lab5-1.o
dizhernakov@dk2n26 ~/work/arch-pc/lab9$
```

Изменяю содержимое файла lab5-2.asm во встроенном редакторе mcsedit (рис. 4.3), чтобы в программе использовались подпрограммы из внешнего файла in out.asm.

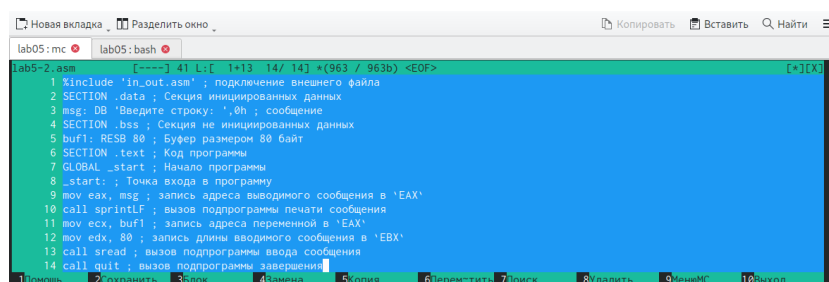


Рис. 4.3: Редактирование файла

Транслирую текст программы файла в объектный файл командой `nasm -f elf lab5-2.asm`. Создался объектный файл `lab5-2.o`. Выполняю компоновку объектного файла с помощью команды `ld -m elf_i386 -o lab5-2 lab5-2.o`. Создался исполняемый файл `lab5-2`. Запускаю исполняемый файл (рис. 4.4).

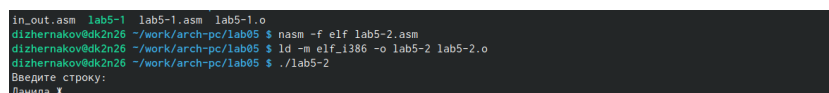


Рис. 4.4: Исполнение файла

Открываю файл `lab5-2.asm` для редактирования в `mcedit` функциональной клавишей `F4`. Изменяю в нем подпрограмму `sprintfLF` на `sprintf`.

Снова транслирую файл, выполняю компоновку созданного объектного файла, запускаю новый исполняемый файл (рис. 4.5).

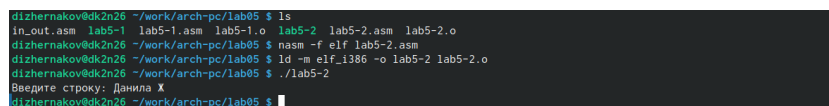
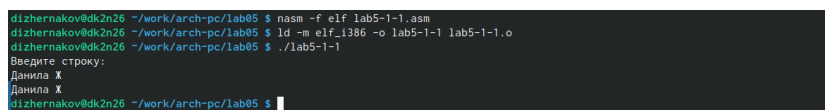


Рис. 4.5: Исполнение файла

Разница между первым исполняемым файлом `lab5-2` и вторым `lab5-2-2` в том, что запуск первого запрашивает ввод с новой строки, а программа, которая выполняется при запуске второго, запрашивает ввод без переноса на новую строку, потому что в этом заключается различие между подпрограммами `sprintfLF` и `sprintf`.

4.3 Выполнение заданий для самостоятельной работы

1. Создаю копию файла lab5-1.asm с именем lab5-1-1.asm С помощью функциональной клавиши F4 открываю созданный файл для редактирования. Изменяю программу так, чтобы кроме вывода приглашения и запроса ввода, она выводила вводимую пользователем строку. Создаю объектный файл lab5-1-1.o, отдаю его на обработку компоновщику, получаю исполняемый файл lab5-1-1, запускаю полученный исполняемый файл. Программа запрашивает ввод, ввожу свои ФИО, далее программа выводит введенные мною данные(рис. 4.6).



```
dizhernakov@dk2n26 ~/work/arch-pc/lab05 $ nasm -f elf lab5-1-1.asm
dizhernakov@dk2n26 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-1-1 lab5-1-1.o
dizhernakov@dk2n26 ~/work/arch-pc/lab05 $ ./lab5-1-1
Введите строку:
Данила X
Данила X
dizhernakov@dk2n26 ~/work/arch-pc/lab05 $
```

Рис. 4.6: Исполнение файла

Код программы из первого пункта:

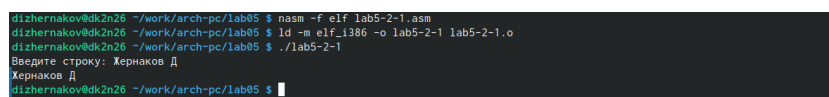
```
SECTION .data ; Секция инициированных данных
msg: DB 'Введите строку:',10
msgLen: EQU $-msg ; Длина переменной 'msg'
SECTION .bss ; Секция не инициированных данных
buf1: RESB 80 ; Буфер размером 80 байт
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу
mov eax,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

```

3. Создаю копию файла lab5-2.asm с именем lab5-2-1.asm. С помощью функциональной клавиши F4 открываю созданный файл для редактирования. Изменяю программу так, чтобы кроме вывода приглашения и запроса ввода, она выводила вводимую пользователем строку
4. Создаю объектный файл lab5-2-1.o, отдаю его на обработку компоновщику, получаю исполняемый файл lab5-2-1, запускаю полученный исполняемый файл. Программа запрашивает ввод без переноса на новую строку, ввожу свои ФИО, далее программа выводит введенные мною данные (рис. 4.7).



```

dizhernakov@dk2n26 ~/work/arch-pc/lab05 $ nasm -f elf lab5-2-1.asm
dizhernakov@dk2n26 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-2-1 lab5-2-1.o
dizhernakov@dk2n26 ~/work/arch-pc/lab05 $ ./lab5-2-1
Введите строку: Хернаков Д
Хернаков Д
dizhernakov@dk2n26 ~/work/arch-pc/lab05 $

```

Рис. 4.7: Редактирование файла

Код программы из третьего пункта:

```

%include 'in_out.asm' SECTION .data ; Секция инициированных 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 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

5 Выводы

При выполнении данной лабораторной работы я приобрёл практические навыки работы в Midnight Commander, а также освоил инструкции языка ассемблера `mov` и `int`.