

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

дисциплина: *Архитектура компьютера*

Студент: Петрова Алевтина Александровна

Группа: НКАбд-05-23

МОСКВА

2023 г.

Содержание

1 Цель работы	3
2 Задание	4
3 Теоретическое введение.....	5
4 Выполнение лабораторной работы.....	7
5 Выводы.....	11
6 Список литературы	12

1 Цель работы

Цель данной лабораторной работы - освоить процедуры компиляции и сборки программ, написанных на ассемблере NASM.

2 Задание

1. Создание программы Hello world!
2. Работа с транслятором NASM
3. Работа с расширенным синтаксисом командной строки NASM
4. Работа с компоновщиком LD
5. Запуск исполняемого файла
6. Выполнение заданий для самостоятельной работы.

3 Теоретическое введение

Основными функциональными элементами любой ЭВМ являются центральный процессор, память и периферийные устройства. Взаимодействие этих устройств осуществляется через общую шину, к которой они подключены. Физически шина представляет собой большое количество проводников, соединяющих устройства друг с другом. В современных компьютерах проводники выполнены в виде электропроводящих дорожек на материнской плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора входят следующие устройства: - арифметико-логическое устройство (АЛУ) — выполняет логические и арифметические действия, необходимые для обработки информации, хранящейся в памяти; - устройство управления (УУ) — обеспечивает управление и контроль всех устройств компьютера; - регистры — сверхбыстрая оперативная память небольшого объёма, входящая в состав процессора, для временного хранения промежуточных результатов выполнения инструкций; регистры процессора делятся на два типа: регистры общего назначения и специальные регистры. Для того, чтобы писать программы на ассемблере, необходимо знать, какие регистры процессора существуют и как их можно использовать. Большинство команд в программах написанных на ассемблере используют регистры в качестве операндов. Практически все команды представляют собой преобразование данных хранящихся в регистрах процессора, это например пересылка данных между регистрами или между регистрами и памятью, преобразование (арифметические или логические операции) данных хранящихся в регистрах. Доступ к регистрам осуществляется не по адресам, как к основной памяти, а по именам. Каждый регистр процессора архитектуры x86 имеет свое название, состоящее из 2 или 3 букв латинского алфавита. В качестве примера приведем названия основных регистров общего назначения (именно эти регистры чаще всего используются при написании программ): - RAX, RCX, RDX, RBX, RSI, RDI — 64-битные - EAX, ECX, EDX, EBX, ESI, EDI — 32-битные - AX, CX, DX, BX, SI, DI — 16-битные - AH, AL, CH, CL, DH, DL, BH, BL — 8-битные

Другим важным узлом ЭВМ является оперативное запоминающее устройство (ОЗУ). ОЗУ — это быстродействующее энергозависимое запоминающее устройство, которое напрямую взаимодействует с узлами процессора, предназначенное для хранения программ и данных, с которыми процессор непосредственно работает в текущий момент. ОЗУ состоит из одинаковых пронумерованных ячеек памяти. Номер ячейки памяти — это адрес хранящихся в ней данных. Периферийные устройства в составе ЭВМ: - устройства внешней памяти, которые предназначены для долговременного хранения больших объёмов данных. - устройства ввода-вывода, которые обеспечивают взаимодействие ЦП с внешней средой.

В основе вычислительного процесса ЭВМ лежит принцип программного управления. Это означает, что компьютер решает поставленную задачу как последовательность действий, записанных в виде программы.

Коды команд представляют собой многоразрядные двоичные комбинации из 0 и 1. В коде машинной команды можно выделить две части: операционную и адресную. В операционной части хранится код команды, которую необходимо выполнить. В

адресной части хранятся данные или адреса данных, которые участвуют в выполнении данной операции. При выполнении каждой команды процессор выполняет определённую последовательность стандартных действий, которая называется командным циклом процессора. Он заключается в следующем: 1. формирование адреса в памяти очередной команды; 2. считывание кода команды из памяти и её дешифрация; 3. выполнение команды; 4. переход к следующей команде.

Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. NASM — это открытый проект ассемблера, версии которого доступны под различные операционные системы и который позволяет получать объектные файлы для этих систем. В NASM используется Intel-синтаксис и поддерживаются инструкции x86-64.

4 Выполнение лабораторной работы

4.1 Создание программы Hello world!

С помощью утилиты `cd` перемещаюсь в каталог, в котором буду работать (рис. 1).

```
[petrovkina1002@fedora ~]$ cd work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab04
[petrovkina1002@fedora lab04]$
```

Рис. 1: Перемещение между директориями

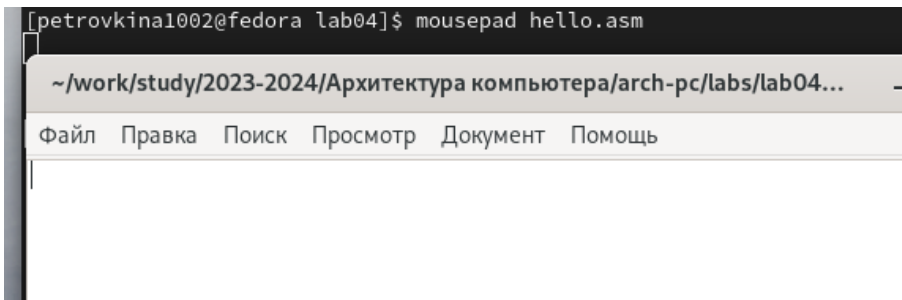
Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты `touch` (рис. 2).

```
[petrovkina1002@fedora lab04]$ touch hello.asm
```

Рис. 2: Создание пустого файла

Открываю созданный файл в текстовом редакторе `mousepad` (рис. 3).

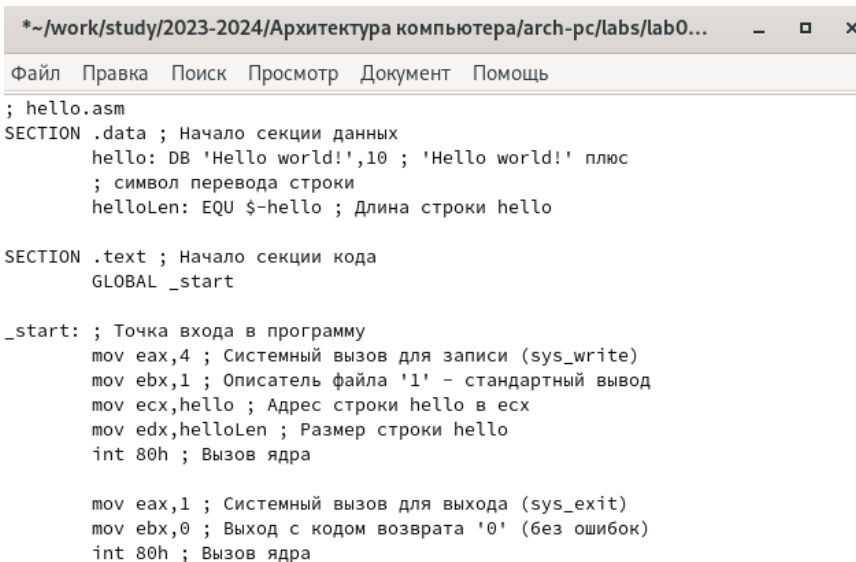
```
[petrovkina1002@fedora lab04]$ mousepad hello.asm
```



The screenshot shows a terminal window with the command `mousepad hello.asm` executed. Below the terminal, a window titled `~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04...` is open. The window has a menu bar with `Файл`, `Правка`, `Поиск`, `Просмотр`, `Документ`, and `Помощь`. The main area of the window is empty, indicating the file `hello.asm` is open but contains no text.

Рис. 3: Открытие файла в текстовом редакторе

Заполняю файл, вставляя в него программу для вывода “Hello word!” (рис. 4).



The screenshot shows the same window as in Figure 3, but now it contains assembly code for a program that prints "Hello world!". The code is as follows:

```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab0...  -  □  x
Файл  Правка  Поиск  Просмотр  Документ  Помощь

; hello.asm
SECTION .data ; Начало секции данных
    hello: DB 'Hello world!',10 ; 'Hello world!' плюс
           ; символ перевода строки
    helloLen: EQU $-hello ; Длина строки hello

SECTION .text ; Начало секции кода
    GLOBAL _start

_start: ; Точка входа в программу
    mov eax,4 ; Системный вызов для записи (sys_write)
    mov ebx,1 ; Описатель файла '1' - стандартный вывод
    mov ecx,hello ; Адрес строки hello в есх
    mov edx,helloLen ; Размер строки hello
    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: Заполнение файла

4.2 Работа с транслятором NASM

Превращаю текст программы для вывода “Hello world!” в объектный код с помощью транслятора NASM, используя команду `nasm -f elf hello.asm`, ключ `-f` указывает транслятору `nasm`, что требуется создать бинарный файл в формате ELF (рис. 5). Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o”.

```
[petrovkina1002@fedora lab04]$ nasm -f elf hello.asm
[petrovkina1002@fedora lab04]$ ls
hello.asm hello.o presentation report
[petrovkina1002@fedora lab04]$
```

Рис. 5: Компиляция текста программы

4.3 Работа с расширенным синтаксисом командной строки NASM

Ввожу команду, которая скомпилирует файл `hello.asm` в файл `obj.o`, при этом в файл будут включены символы для отладки (ключ `-g`), также с помощью ключа `-l` будет создан файл листинга `list.lst` (рис. 6). Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```
[petrovkina1002@fedora lab04]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
[petrovkina1002@fedora lab04]$ ls
hello.asm hello.o list.lst obj.o presentation report
[petrovkina1002@fedora lab04]$
```

Рис. 6: Компиляция текста программы

4.4 Работа с компоновщиком LD

Передаю объектный файл `hello.o` на обработку компоновщику `LD`, чтобы получить исполняемый файл `hello` (рис. 7). Ключ `-o` задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```
[petrovkina1002@fedora lab04]$ ld -m elf_i386 hello.o -o hello
[petrovkina1002@fedora lab04]$ ls
hello hello.asm hello.o list.lst obj.o presentation report
[petrovkina1002@fedora lab04]$
```

Рис. 7: Передача объектного файла на обработку компоновщику

Выполняю следующую команду (рис. 8). Исполняемый файл будет иметь имя `main`, т.к. после ключа `-o` было задано значение `main`. Объектный файл, из которого собран этот исполняемый файл, имеет имя `obj.o`

```
[petrovkina1002@fedora lab04]$ ld -m elf_i386 obj.o -o main
[petrovkina1002@fedora lab04]$ ls
hello hello.asm hello.o list.lst main obj.o presentation report
[petrovkina1002@fedora lab04]$
```

Рис. 8: Передача объектного файла на обработку компоновщику

4.5 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello (рис. 9).

```
[petrovkina1002@fedora lab04]$ ./hello
Hello world!
```

Рис. 9: Запуск исполняемого файла

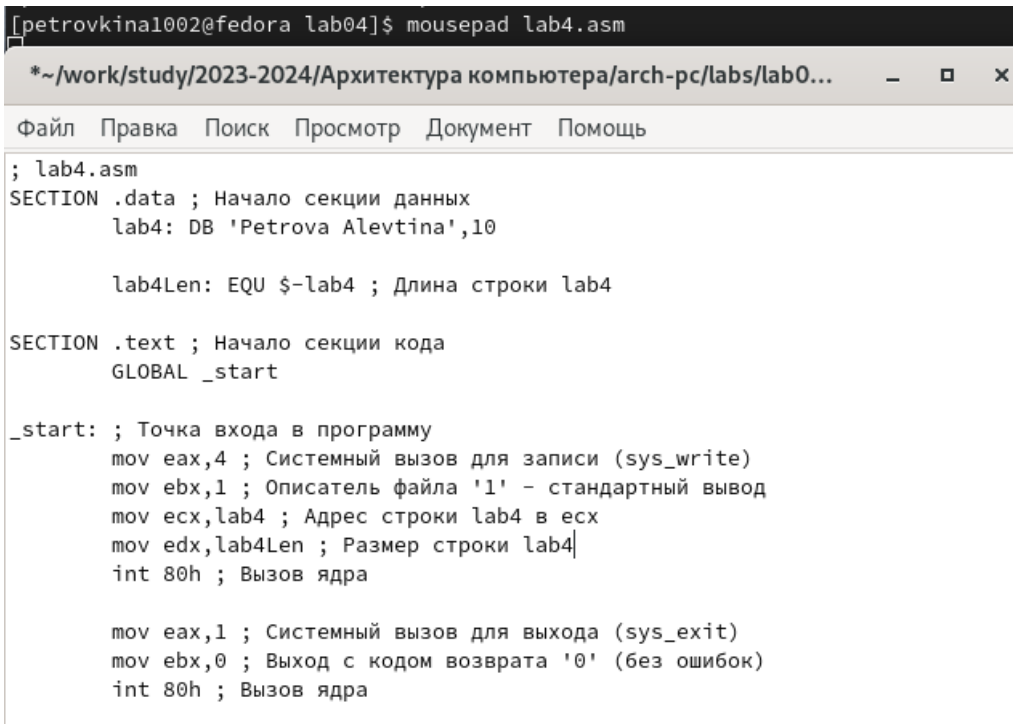
4.6 Выполнение заданий для самостоятельной работы.

С помощью утилиты `cp` создаю в текущем каталоге копию файла `hello.asm` с именем `lab4.asm` (рис. 10).

```
[petrovkina1002@fedora lab04]$ cp hello.asm lab4.asm
```

Рис. 10: Создание копии файла

С помощью текстового редактора `mousepad` открываю файл `lab4.asm` и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 11).



```
[petrovkina1002@fedora lab04]$ mousepad lab4.asm
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab0...
Файл Правка Поиск Просмотр Документ Помощь

; lab4.asm
SECTION .data ; Начало секции данных
    lab4: DB 'Petrova Alevtina',10

    lab4Len: EQU $-lab4 ; Длина строки lab4

SECTION .text ; Начало секции кода
    GLOBAL _start

_start: ; Точка входа в программу
    mov eax,4 ; Системный вызов для записи (sys_write)
    mov ebx,1 ; Описатель файла '1' - стандартный вывод
    mov ecx,lab4 ; Адрес строки lab4 в ecx
    mov edx,lab4Len ; Размер строки lab4
    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. 11: Изменение программы

Компилирую текст программы в объектный файл (рис. 12). Проверяю с помощью утилиты `ls`, что файл `lab4.o` создан.

```
[petrovkina1002@fedora lab04]$ nasm -f elf lab4.asm
[petrovkina1002@fedora lab04]$ ls
hello  hello.asm  hello.o  lab4.asm  lab4.o  list.lst  main  obj.o  presentation  report
[petrovkina1002@fedora lab04]$
```

Рис. 12: Компиляция текста программы

Передаю объектный файл `lab4.o` на обработку компоновщику `LD`, чтобы получить исполняемый файл `lab4` (рис. 13).

```
[petrovkina1002@fedora lab04]$ ld -m elf_i386 lab4.o -o lab4
[petrovkina1002@fedora lab04]$ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o presentation report
[petrovkina1002@fedora lab04]$
```

Рис. 13: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab4, на экран действительно выводятся мои имя и фамилия (рис. 14).

```
[petrovkina1002@fedora lab04]$ ./lab4
Petrova Alevtina
```

Рис. 14: Запуск исполняемого файла

Удаляю лишние файлы в текущем каталоге с помощью утилиты rm (рис. 16).

```
[petrovkina1002@fedora lab04]$ rm hello hello.o lab4 lab4.o list.lst main obj.o
[petrovkina1002@fedora lab04]$ ls
hello.asm lab4.asm presentation report
[petrovkina1002@fedora lab04]$
```

Рис. 15: Удаление лишних файлов в текущем каталоге

С помощью команд git add . и git commit добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №4 (рис. 17).

```
[petrovkina1002@fedora lab04]$ git add .
[petrovkina1002@fedora lab04]$ git commit -m "Add files for lab04"
[master 6e7c7e1] Add files for lab04
2 files changed, 38 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
[petrovkina1002@fedora lab04]$
```

Рис. 16: Добавление файлов на GitHub

Отправляю файлы на сервер с помощью команды git push (рис. 18).

```
[petrovkina1002@fedora lab04]$ git push
Перечисление объектов: 9, готово.
Подсчет объектов: 100% (9/9), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (6/6), готово.
Запись объектов: 100% (6/6), 1.03 КиБ | 352.00 КиБ/с, готово.
Всего 6 (изменений 3), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
```

Рис. 17: Отправка файлов

5 Выводы

При выполнении данной лабораторной работы я освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM.

6 Список литературы

1. https://esystem.rudn.ru/pluginfile.php/1584628/mod_resource/content/1/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%965.pdf