

Отчёт по лабораторной работе №4

Дисциплина: Архитектура Компьютера

Егор Витальевич Кузьмин

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
5	Выводы	12
	Список литературы	13

Список иллюстраций

4.1	Создание каталога	8
4.2	Перемещение	8
4.3	открытие файла	8
4.4	Ввод текста	9
4.5	Компиляция текста программы	9
4.6	Компиляция текста программы	9
4.7	Передача объектного файла на обработку компоновщику	10
4.8	Запуск исполняемого файла	10
4.9	Создание копии файла и его открытие, редактирование	10
4.10	Компиляция, передача компоновщику	11
4.11	Отправка файлов	11

1 Цель работы

Целью данной работы является приобретение практического опыта работы с программами, написанными на ассемблере NASM, а именно - освоение процедур компиляций и сборки.

2 Задание

- 0. Общее ознакомление с NASM
- 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

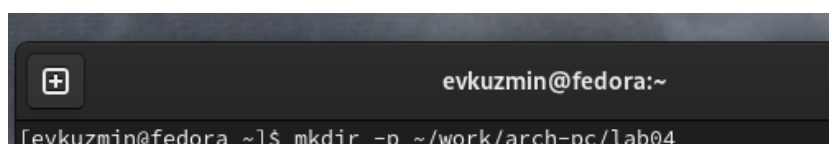
3 Теоретическое введение

Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. Можно считать, что он больше любых других языков приближен к архитектуре ЭВМ и её аппаратным возможностям, что позволяет получить к ним более полный доступ, нежели в языках высокого уровня, таких как C/C++, Perl, Python и пр. Но получить полный доступ к ресурсам компьютера в современных архитектурах нельзя, самым низким уровнем работы прикладной программы является обращение напрямую к ядру операционной системы. Именно на этом уровне и работают программы, написанные на ассемблере. Но в отличие от языков высокого уровня ассемблерная программа содержит только тот код, который ввёл программист. Таким образом язык ассемблера — это язык, с помощью которого понятным для человека образом пишутся команды для процессора. Следует отметить, что процессор понимает не команды ассемблера, а последовательности из нулей и единиц — машинные коды. До появления языков ассемблера программистам приходилось писать программы, используя только лишь машинные коды, которые были крайне сложны для запоминания, так как представляли собой числа, записанные в двоичной или шестнадцатеричной системе счисления. Преобразование или трансляция команд с языка ассемблера в исполняемый машинный код осуществляется специальной программой — транслятором. Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, ибо транслятор просто переводит мнемонические обозначения команд в последовательности бит (нулей и единиц). Используемые

мнемоники обычно одинаковы для всех процессоров одной архитектуры или семейства архитектур (среди широко известных — мнемоники процессоров и контроллеров x86, ARM, SPARC, PowerPC, M68k). Таким образом для каждой архитектуры существует свой ассемблер и, соответственно, свой язык ассемблера. Наиболее распространёнными ассемблерами для архитектуры x86 являются: 1) Для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM). 2) Для GNU/Linux: gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис. Для записи команд в NASM используются: 1) Мнемокод — непосредственно мнемоника инструкции процессору, которая является обязательной частью команды. 2) Операнды — числа, данные, адреса регистров или адреса оперативной памяти. 3) Метка — идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. (Метка перед командой связана с адресом данной команды). Допустимыми символами в метках являются буквы, цифры, а также следующие символы: `,`, `$`, `#`, `@`, `~`, `.` и `?`. *Начинаться метка или идентификатор могут с буквы, `.`, и `?`.* Перед идентификаторами, которые пишутся как зарезервированные слова, нужно писать `$`, чтобы компилятор трактовал его верно (так называемое экранирование). Максимальная длина идентификатора составляет 4095 символов. Программа на языке ассемблера также может содержать директивы — инструкции, не переводящиеся непосредственно в машинные команды, а управляющие работой транслятора. Например, директивы используются для определения данных (констант и переменных) и обычно пишутся большими буквами.

4 Выполнение лабораторной работы

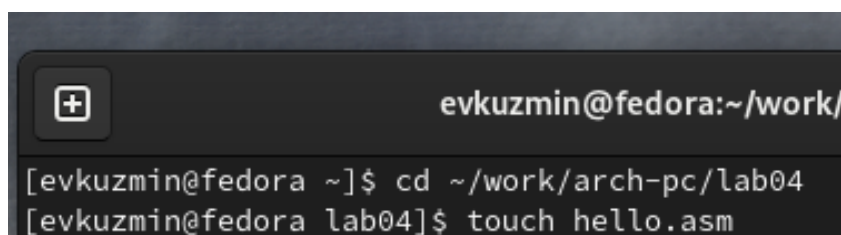
4.1) Программа Hello World! Создаю каталог для работы с программами на языке ассемблера NASM. (рис. 4.1).

A terminal window with a dark background. The title bar shows a window icon and the text 'evkuzmin@fedora:~'. The command prompt shows '[evkuzmin@fedora ~]\$' followed by the command 'mkdir -p ~/work/arch-pc/lab04'.

```
evkuzmin@fedora:~  
[evkuzmin@fedora ~]$ mkdir -p ~/work/arch-pc/lab04
```

Рис. 4.1: Создание каталога

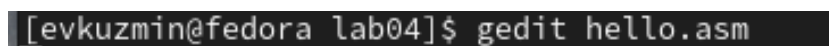
Перехожу в каталог и создаю текстовый файл hello.asm. (рис. 4.2).

A terminal window with a dark background. The title bar shows a window icon and the text 'evkuzmin@fedora:~/work/'. The command prompt shows '[evkuzmin@fedora ~]\$' followed by 'cd ~/work/arch-pc/lab04', and then '[evkuzmin@fedora lab04]\$' followed by 'touch hello.asm'.

```
evkuzmin@fedora:~/work/  
[evkuzmin@fedora ~]$ cd ~/work/arch-pc/lab04  
[evkuzmin@fedora lab04]$ touch hello.asm
```

Рис. 4.2: Перемещение

Открытие файла. (рис. 4.3).

A terminal window with a dark background. The command prompt shows '[evkuzmin@fedora lab04]\$' followed by 'gedit hello.asm'.

```
[evkuzmin@fedora lab04]$ gedit hello.asm
```

Рис. 4.3: открытие файла

Ввожу нужный текст. (рис. 4.4).

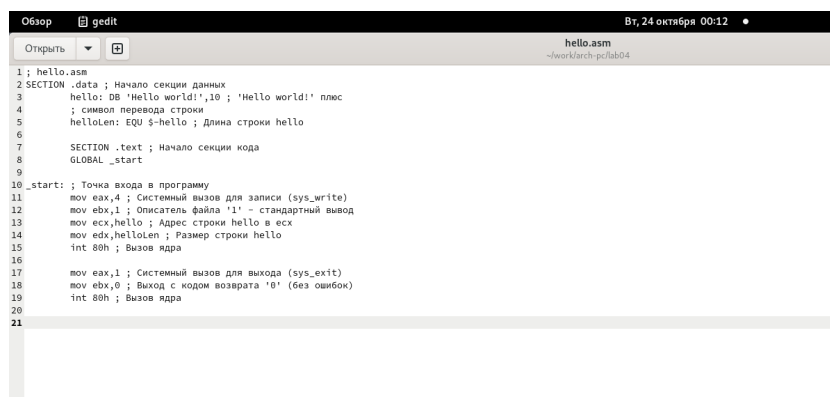


Рис. 4.4: Ввод текста

4.2) Транслятор NASM

NASM превращает текст программы в объектный код. Выполним компиляцию приведённого выше текста программы “Hello World”. Сделаем проверку. (рис. 4.5).

```

[evkuzmin@fedora lab04]$ nasm -f elf hello.asm
[evkuzmin@fedora lab04]$ ls
hello.asm  hello.o

```

Рис. 4.5: Компиляция текста программы

4.3) Расширенный синтаксис 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. Проверяю правильность выполнения команды. (рис. 4.6).

```

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

```

Рис. 4.6: Компиляция текста программы

4.4) Работа с компоновщиком LD

Передаю объектный файл hello.o на обработку компоновщику LD, чтобы получить исполняемый файл hello. Ключ -o задает имя создаваемого исполняемого файла. Выполняю ту же самую команду со значением main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o (рис. 4.7).

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

Рис. 4.7: Передача объектного файла на обработку компоновщику

4.5) Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello. (рис. 4.8).

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

Рис. 4.8: 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. С помощью текстового редактора открываю файл lab4.asm и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 4.9).

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

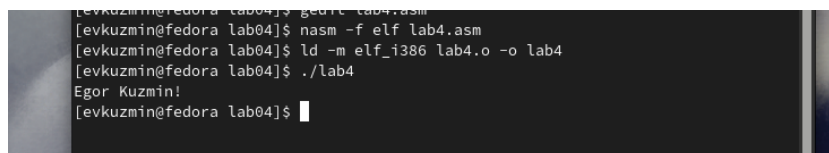
*lab4.asm
~/work/arch-pc/lab04

1; hello.asm
2 SECTION .data ; Начало секции данных
3     lab4: DB "Egor Kuzmin",10 ; 'Hello world!' нпмс
4     ; символ перевода строки
5     helloLen: EQU $-hello ; Длина строки hello
6
```

Рис. 4.9: Создание копии файла и его открытие, редактирование

Компилирую текст программы в объектный файл, передаю его компоновщику.

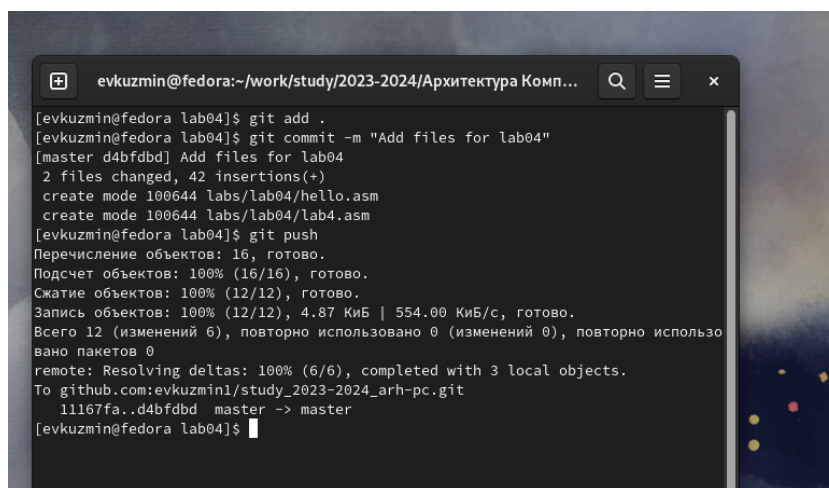
Запускаю исполняемый файл lab4, на экран действительно выводятся мои имя и фамилия. (рис. 4.10)



```
[evkuzmin@fedora lab04]$ gcc -c lab4.asm
[evkuzmin@fedora lab04]$ nasm -f elf lab4.asm
[evkuzmin@fedora lab04]$ ld -m elf_i386 lab4.o -o lab4
[evkuzmin@fedora lab04]$ ./lab4
Egor Kuzmin!
[evkuzmin@fedora lab04]$
```

Рис. 4.10: Компиляция, передача компоновщику

Вручную копирую файлы hello.asm и lab4.asm в основной репозиторий курса. Отправляю все файлы на github. (рис. 4.11)



```
evkuzmin@fedora: ~/work/study/2023-2024/Архитектура Комп...
[evkuzmin@fedora lab04]$ git add .
[evkuzmin@fedora lab04]$ git commit -m "Add files for lab04"
[master d4bfdbd] Add files for lab04
2 files changed, 42 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
[evkuzmin@fedora lab04]$ git push
Перечисление объектов: 16, готово.
Подсчет объектов: 100% (16/16), готово.
Сжатие объектов: 100% (12/12), готово.
Запись объектов: 100% (12/12), 4.87 КиБ | 554.00 КиБ/с, готово.
Всего 12 (изменений 6), повторно использовано 0 (изменений 0), повторно использо
вано пакетов 0
remote: Resolving deltas: 100% (6/6), completed with 3 local objects.
To github.com:evkuzmin1/study_2023-2024_arh-pc.git
 11167fa..d4bfdbd master -> master
[evkuzmin@fedora lab04]$
```

Рис. 4.11: Отправка файлов

5 Выводы

При выполнении лабораторной работы я обрёл практический опыт работы с программами, написанными на ассемблере NASM, конкретнее - освоил процедуры компиляций и сборки.

Список литературы

Архитектура ЭВМ