

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Отчет по лабораторной работе №4

**“Создание и процесс обработки программ
на языке ассемблера NASM”**

Студент: Богомолова Полина Петровна

Группа: НКАбд-01-25

Москва 2025

Оглавление

Цель работы.....	2
Теоретическое введение.....	3
Основные принципы работы компьютера.....	3
Рис. 1	4
Ассемблер и язык ассемблера	4
Процесс создания и обработки программы на языке ассемблера.....	5
Рис. 2	6
Порядок выполнения лабораторной работы.....	7
Программа Hello World!	7
Рис. 3	7
Рис. 4	8
Транслятор NASM.....	8
Рис. 5	8
Рис. 6	8
Расширенный синтаксис командной строки NASM	9
Рис. 7	9
Компоновщик ld.....	9
Рис. 8	9
Рис. 9	9
Запуск исполняемого файла	10
Рис. 10	10
Выполнение заданий для самостоятельной работы	10
Рис. 11	10
Рис. 12	11
Рис. 13	11
Рис. 14	12
Рис. 15	12
Рис. 16	12
Рис. 17	13
Вывод.....	14
Список литературы.....	15

Цель работы

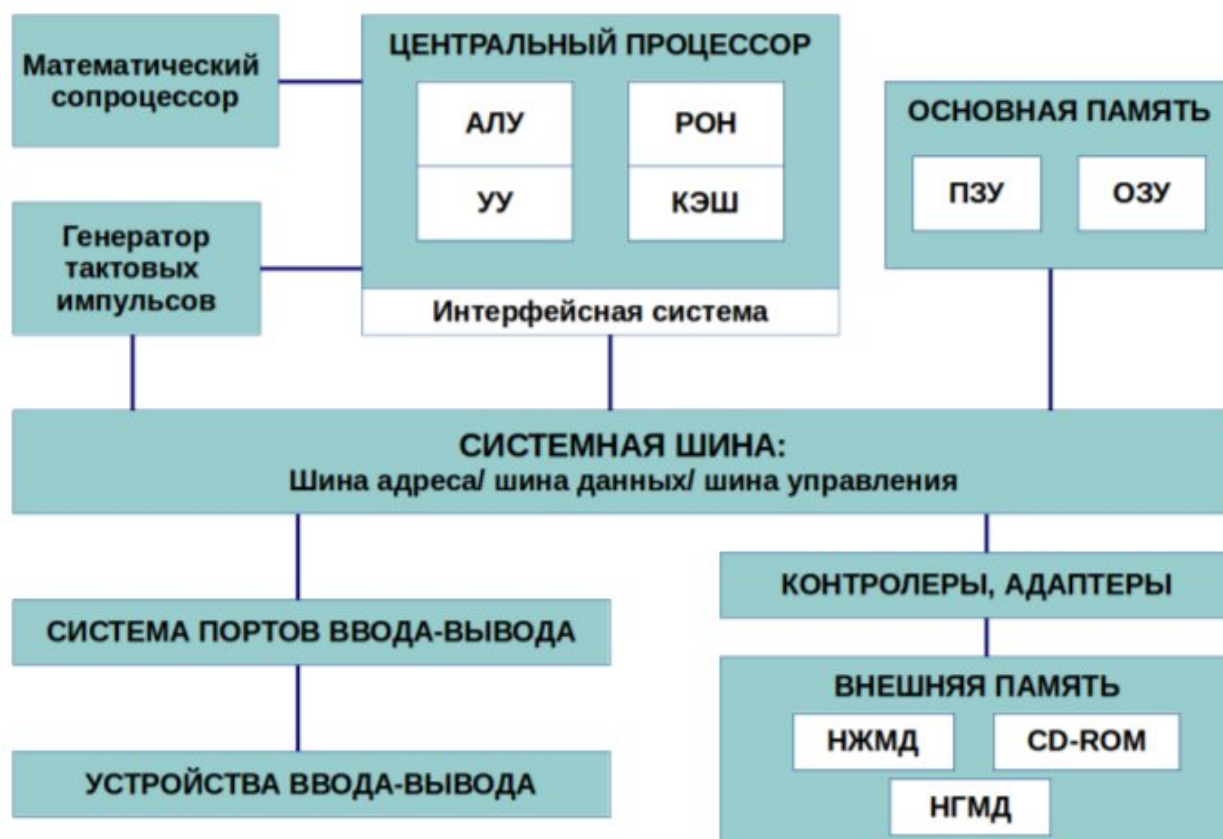
Освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

Теоретическое введение

Основные принципы работы компьютера

Основными функциональными элементами любой электронно-вычислительной машины (ЭВМ) являются центральный процессор, память и периферийные устройства (рис. 1). 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 (системной) плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора (ЦП) 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.

Рис. 1



Ассемблер и язык ассемблера

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. Заметим, что 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 — Ассемблер. 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.

[метка:] мнемокод [операнд {, операнд}] [; комментарий]

Здесь мнемокод — непосредственно мнемоника инструкции процессору, которая является обязательной частью команды. Операндами могут быть числа, данные, адреса регистров или адреса оперативной памяти. Метка — это идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. Т.о. 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 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.

Процесс создания и обработки программы на языке ассемблера

Процесс создания ассемблерной программы можно изобразить в виде следующей схемы (рис. 2).

Рис. 2



В процессе создания ассемблерной программы можно выделить четыре шага:

- Набор текста программы в текстовом редакторе и сохранение её в отдельном файле. Каждый файл имеет свой тип (или расширение), который определяет назначение файла. Файлы с исходным текстом программ на языке ассемблера имеют тип `asm`.
- Трансляция — преобразование с помощью транслятора, например `nasm`, текста программы в машинный код, называемый объектным. На данном этапе также может быть получен листинг программы, содержащий кроме текста программы различную дополнительную информацию, созданную транслятором. Тип объектного файла — `o`, файла листинга — `lst`.
- Компоновка или линковка — этап обработки объектного кода компоновщиком (`ld`), который принимает на вход объектные файлы и собирает по ним исполняемый

файл. Исполняемый файл обычно не имеет расширения. Кроме того, можно получить файл карты загрузки программы в ОЗУ, имеющий расширение `map`.

- **Запуск программы.** Конечной целью является работоспособный исполняемый файл. Ошибки на предыдущих этапах могут привести к некорректной работе программы, поэтому может присутствовать этап отладки программы при помощи специальной программы — отладчика. При нахождении ошибки необходимо провести коррекцию программы, начиная с первого шага.

Порядок выполнения лабораторной работы

Программа Hello World!

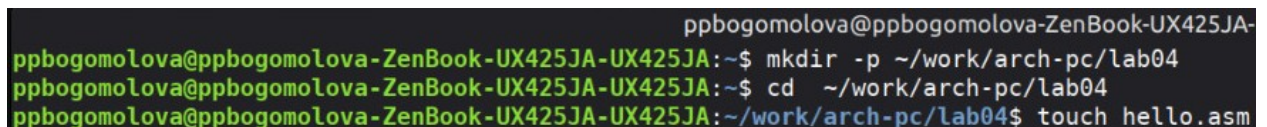
Создадим каталог для работы с программами на языке ассемблера NASM с помощью команды `mkdir -p`.

Перейдем в созданный каталог с помощью команды `cd`.

Создадим файл `'hello.asm'` с помощью команды `touch`.

Результат представлен на рисунке 3.

Рис. 3

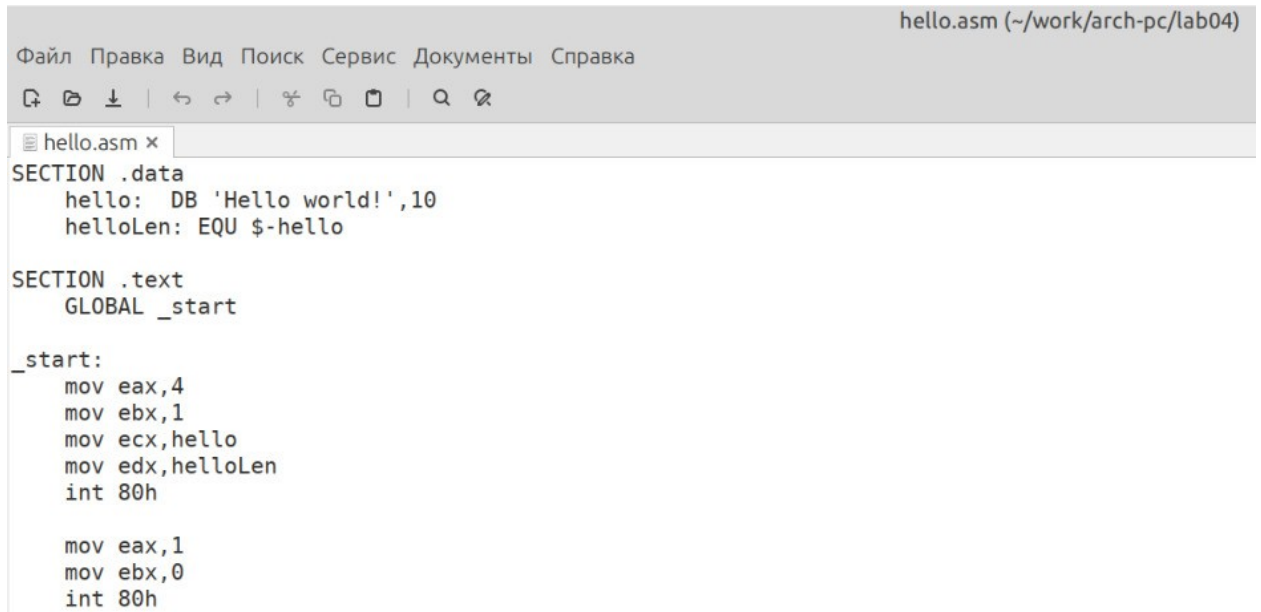


```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~$ mkdir -p ~/work/arch-pc/lab04
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~$ cd ~/work/arch-pc/lab04
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$ touch hello.asm
```

Откроем файл в текстовом редакторе `gedit`. Введем в файл текст.

Результат представлен на рисунке 4.

Рис. 4



```
hello.asm (~/work/arch-pc/lab04)
Файл  Правка  Вид  Поиск  Сервис  Документы  Справка
[Icons]
hello.asm x
SECTION .data
    hello: DB 'Hello world!',10
    helloLen: EQU $-hello

SECTION .text
    GLOBAL _start

_start:
    mov eax,4
    mov ebx,1
    mov ecx,hello
    mov edx,helloLen
    int 80h

    mov eax,1
    mov ebx,0
    int 80h
```

Транслятор NASM

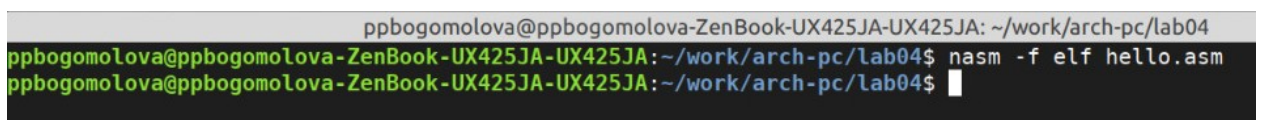
Превратим текст программы в объектный код с помощью команды `nasm -f elf`.

С помощью команды `ls` проверим, что объектный файл был создан.

Созданный объектный файл имеет имя 'hello.o'.

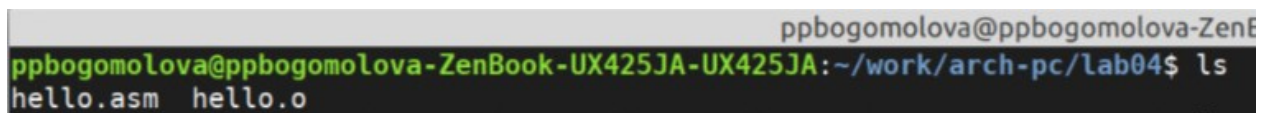
Результат представлен на рисунках 5-6.

Рис. 5



```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab04
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$ nasm -f elf hello.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$
```

Рис. 6



```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$ ls
hello.asm  hello.o
```

Расширенный синтаксис командной строки NASM

Выполним команду `nasm -o obj.o -f elf -g -l list.lst hello.asm`.

Данная команда скомпилирует исходный файл `hello.asm` в `obj.o` (опция `-o` позволяет задать имя объектного файла, в данном случае `obj.o`), при этом формат выходного файла будет `elf`, и в него будут включены символы для отладки (опция `-g`), кроме того, будет создан файл листинга `list.lst` (опция `-l`). С помощью команды `ls` проверим, что файлы созданы.

Результат представлен на рисунке 7.

Рис. 7

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab04
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst hello.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$ ls
hello.asm hello.o list.lst obj.o
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$
```

Компоновщик ld

Чтобы получить исполняемую программу, объектный файл передадим на обработку компоновщику. Для этого будем использовать команду `ld -m elf_i386 hello.o -o hello`.

С помощью команды `ls` проверим, что исполняемый файл `hello` был создан.

Результат представлен на рисунке 8.

Рис. 8

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab04
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o list.lst obj.o
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$
```

Выполним команду `ld -m elf_i386 obj.o -o main`.

Исполняемый файл будет называться `'main'`, объектный файл `'obj.o'`.

Результат представлен на рисунке 9.

Рис. 9

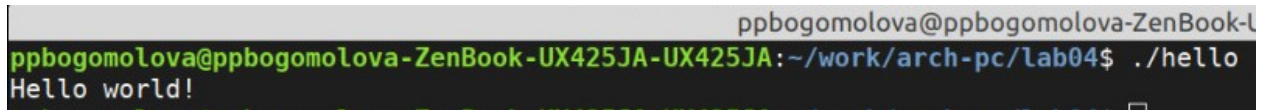
```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o list.lst main obj.o
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab04$
```

Запуск исполняемого файла

Чтобы запустить на выполнение созданный исполняемый файл, находящийся в текущем каталоге, наберем в командной строке команду `./hello`.

Результат представлен на рисунке 10.

Рис. 10



```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab04$ ./hello
Hello world!
```

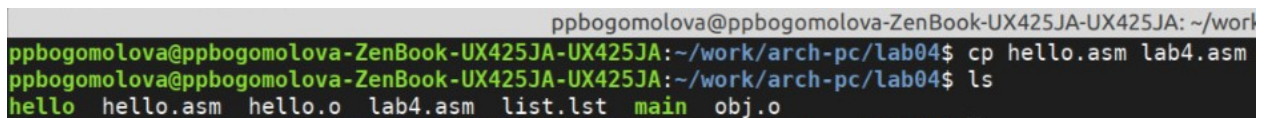
Выполнение заданий для самостоятельной работы

1. В каталоге `~/work/arch-pc/lab04` с помощью команды `cp` создайте копию файла `hello.asm` с именем `lab4.asm`

С помощью команды `cp` создадим копию, с помощью команды `ls` проверим, что копия действительно создана.

Решение представлено на рисунке 11.

Рис. 11



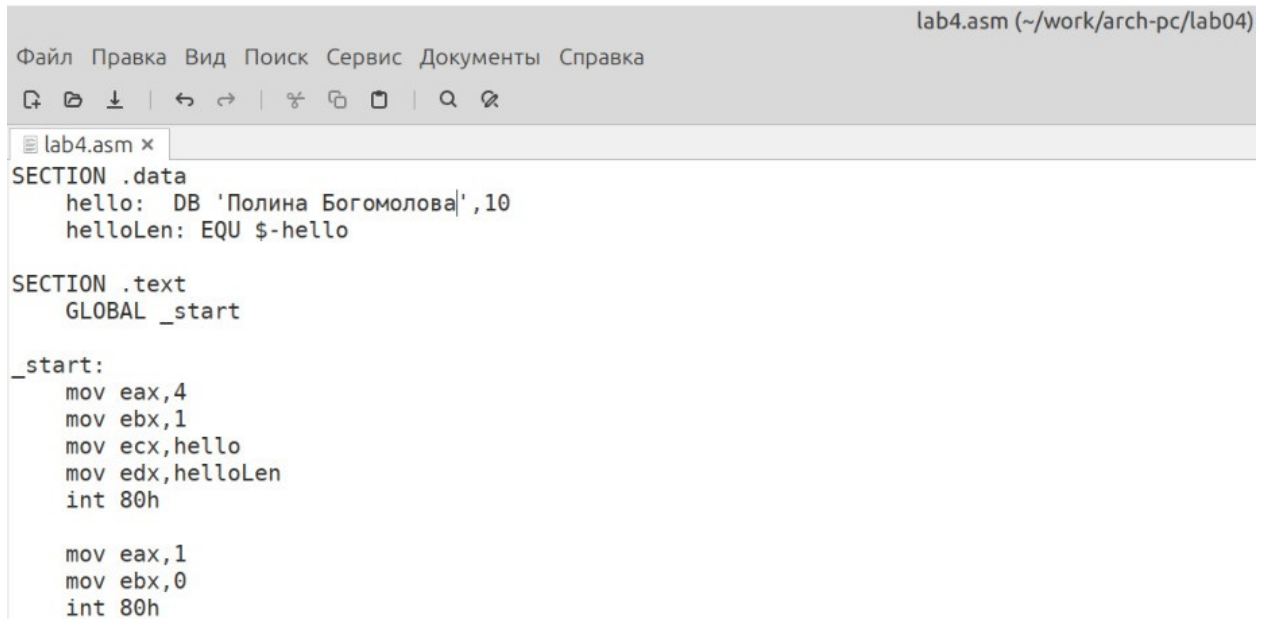
```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab04$ cp hello.asm lab4.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab4.asm  list.lst  main  obj.o
```

2. С помощью любого текстового редактора внесите изменения в текст программы в файле `lab4.asm` так, чтобы вместо `Hello world!` на экран выводилась строка с вашими фамилией и именем.

В тексте файла заменим `'Hello world!'` на `'Полина Богомолова'`.

Решение представлено на рисунке 12.

Рис. 12



```
lab4.asm (~/.work/arch-pc/lab04)
Файл Правка Вид Поиск Сервис Документы Справка
lab4.asm x
SECTION .data
    hello: DB 'Полина Богомолова',10
    helloLen: EQU $-hello

SECTION .text
    GLOBAL _start

_start:
    mov eax,4
    mov ebx,1
    mov ecx,hello
    mov edx,helloLen
    int 80h

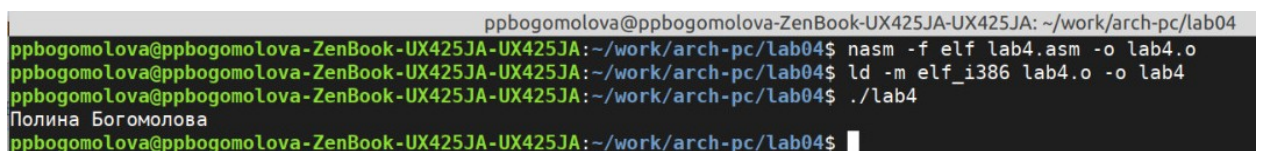
    mov eax,1
    mov ebx,0
    int 80h
```

3. Оттранслируйте полученный текст программы lab4.asm в объектный файл. Выполните компоновку объектного файла и запустите получившийся исполняемый файл

Превратим текст программы в объектный код с помощью команды `nasm -f elf`. Чтобы получить исполняемую программу, объектный файл передадим на обработку компоновщику. Для этого будем использовать команду `ld -m elf_i386 lab4.o -o lab4`. Чтобы запустить на выполнение созданный исполняемый файл, находящийся в текущем каталоге, наберем в командной строке команду `./lab4`. Заметим, что при запуске файла на экран выводятся мои имя и фамилия.

Решение представлено на рисунке 13.

Рис. 13



```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/.work/arch-pc/lab04
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/.work/arch-pc/lab04$ nasm -f elf lab4.asm -o lab4.o
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/.work/arch-pc/lab04$ ld -m elf_i386 lab4.o -o lab4
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/.work/arch-pc/lab04$ ./lab4
Полина Богомолова
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/.work/arch-pc/lab04$
```

4. Скопируйте файлы hello.asm и lab4.asm в Ваш локальный репозиторий в каталог `~/work/study/2023-2024/"Архитектура компьютера"/arch-`

pc/labs/lab04/. Загрузите файлы на Github.

Просмотрим состояние репозитория с помощью команды `git status`. С помощью команды `git add` добавляем все новые файлы. Создаем коммит - 'снимок' текущего состояния добавленных файлов. Отправляем локальные коммиты на удаленный репозиторий.

Решение представлено на рисунках 14-17.

Рис. 14

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/study/2025-2026/arh/study_2025-2026_arh-pc$ git status
Текущая ветка: master
Эта ветка соответствует «origin/master».

Изменения, которые будут включены в коммит:
  (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
    новый файл:   labs/lab04/hello.asm
    новый файл:   labs/lab04/lab4.asm
```

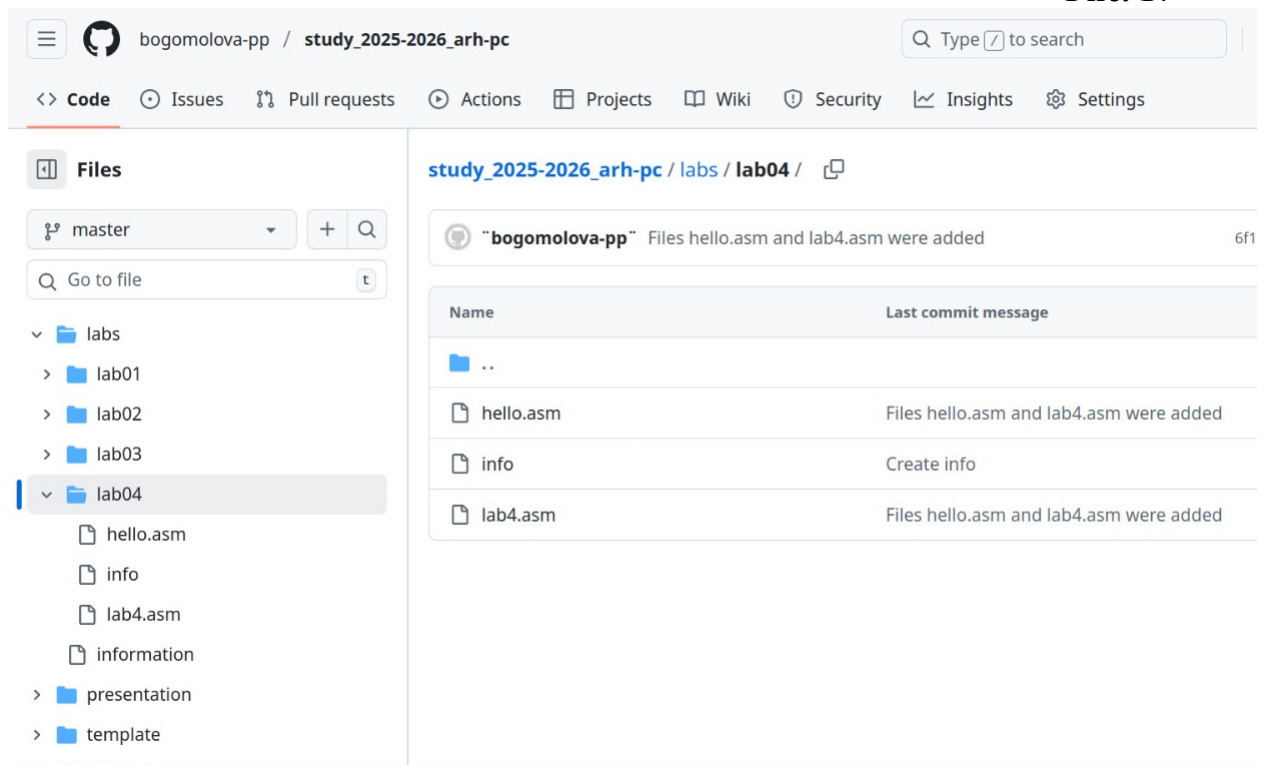
Рис. 15

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/study/2025-2026/arh/study_2025-2026_arh-pc$ git add .
```

Рис. 16

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/study/2025-2026/arh/study_2025-2026_arh-pc$ git commit -m "Files hello.asm and lab4.asm were added"
[master 6f1bc0a] Files hello.asm and lab4.asm were added
 2 files changed, 34 insertions(+)
 create mode 100644 labs/lab04/hello.asm
 create mode 100644 labs/lab04/lab4.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/study/2025-2026/arh/study_2025-2026_arh-pc$ git push --set-upstream origin master
Перечисление объектов: 7, готово.
Подсчет объектов: 100% (7/7), готово.
При сжатии изменений используется до 8 потоков
Сжатие объектов: 100% (4/4), готово.
Запись объектов: 100% (4/4), 449 байтов | 449.00 КиБ/с, готово.
Всего 4 (изменений 2), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:bogomolova-pp/study_2025-2026_arh-pc.git
 83f984d..6f1bc0a  master -> master
branch 'master' set up to track 'origin/master'.
```

Рис. 17



Ссылка на мой репозиторий в Github:

https://github.com/bogomolova-pp/study_2025-2026_arh-pc

Вывод

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

Список литературы

1. Лабораторная работа №4 – Демидова А.В