

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

**Факультет физико-математических и естественных наук
Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4**

дисциплина: *Архитектура Вычислительных Систем*

Студент: Сидоренко Максим Алексеевич

Группа: НБИбд-02-22

МОСКВА

2022 г.

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

Лабораторная работа

Ход работы:

1) Программа hello world! 3.4.

- Рассмотрим пример простой программы на языке ассемблера NASM. Традиционно первая программа выводит приветственное сообщение Hello world! на экран.
- Создадим каталог для работы с программами на языке ассемблера NASM:

```
masidorenko@masidorenko: ~  
masidorenko@masidorenko:~$ mkdir -p work/arch-pc/lab04  
masidorenko@masidorenko:~$
```

- Перейдем в созданный каталог

```
masidorenko@masidorenko:~$ cd ~/work/arch-pc/lab04  
masidorenko@masidorenko:~/work/arch-pc/lab04$
```

- Создадим текстовый файл с именем hello.asm

```
masidorenko@masidorenko:~/work/arch-pc/lab04$ touch hello.asm
```

- Откроем этот файл с помощью любого текстового редактора, например, gedit

```
masidorenko@masidorenko:~/work/arch-pc/lab04$ gedit hello.asm  
[
```

- Введем в него следующий текст:

```

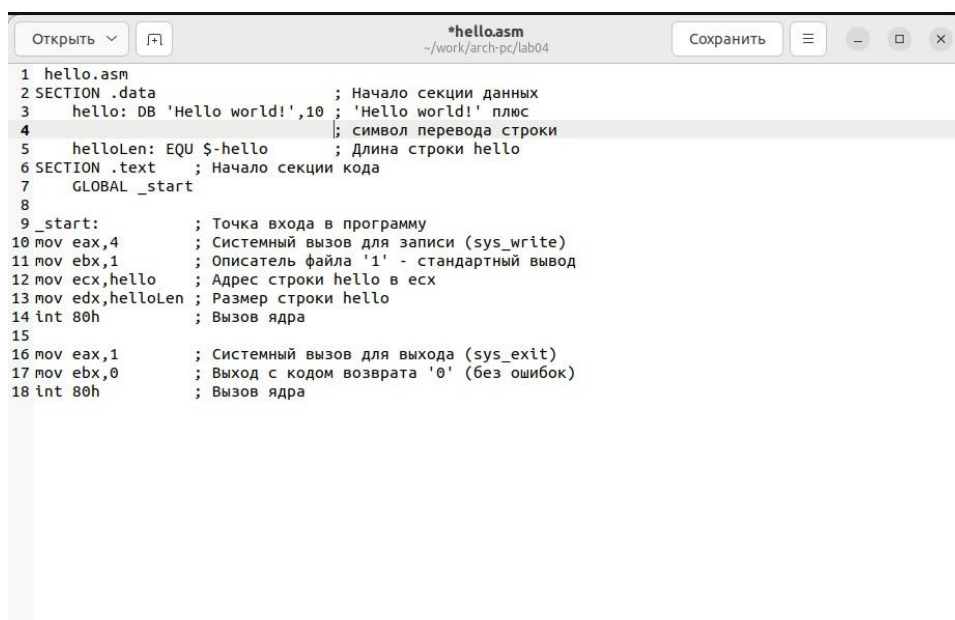
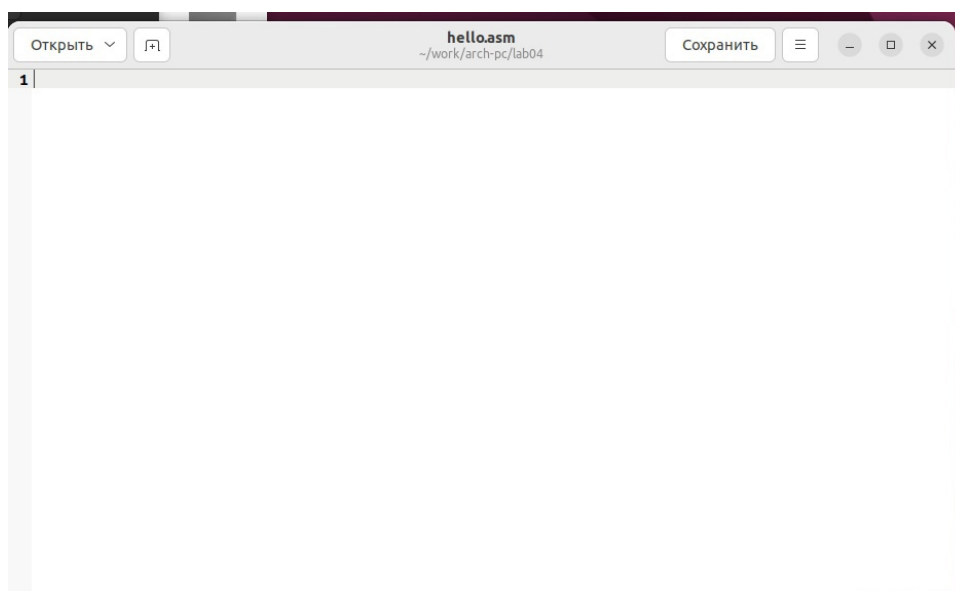
; hello.asm
SECTION .data                ; Начало секции данных
    hello:    DB 'Hello world!',10 ; 'Hello world!' плюс
                                           ; символ перевода строки
    helloLen: EQU $-hello      ; Длина строки hello

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

```



- В отличие от многих современных высокоуровневых языков программирования, в ассемблерной программе каждая команда располагается на отдельной строке. Размещение нескольких команд на одной строке недопустимо. Синтаксис ассемблера

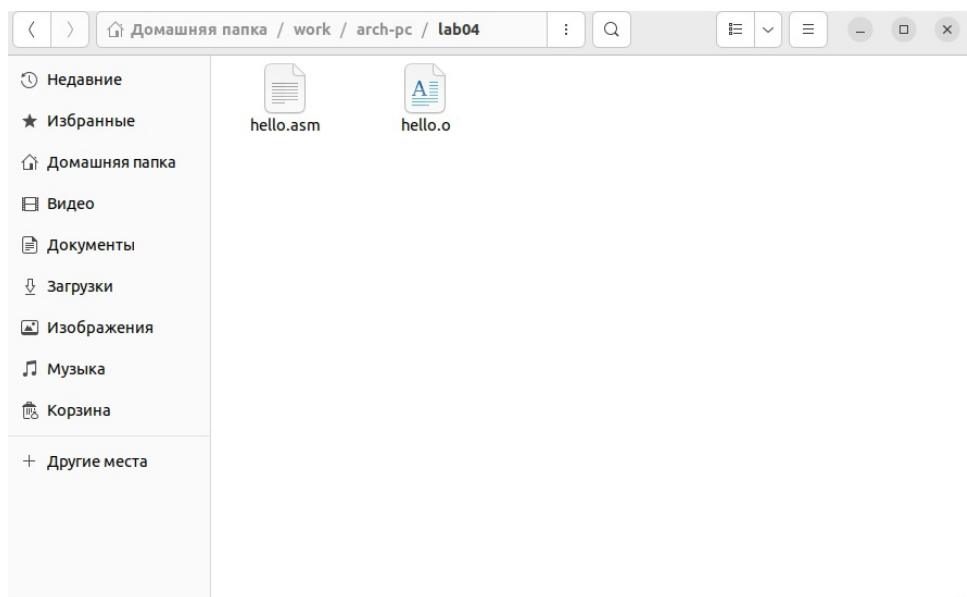
NASM является чувствительным к регистру, т.е. есть разница между большими и малыми буквами

2) Транслятор NASM 4.3.2.

- NASM превращает текст программы в объектный код. Например, для компиляции приведённого выше текста программы «Hello World» необходимо написать:

```
nasidorenko@nasidorenko: ~/work/arch-pc/lab04$ nasm -f elf hello.asm
hello.asm:1: warning: label alone on a line without a colon might be in error [-w+label-orphan]
```

- Если текст программы набран без ошибок, то транслятор преобразует текст программы из файла hello.asm в объектный код, который запишется в файл hello.o. Таким образом, имена всех файлов получаются из имени входного файла и расширения по умолчанию. При наличии ошибок объектный файл не создаётся, а после запуска транслятора появятся сообщения об ошибках или предупреждения.
- С помощью команды ls проверим, что объектный файл был создан.
- **(Вопрос: Какое имя имеет объектный файл?. Ответ: hello.o)**
- NASM не запускают без параметров. Ключ -f указывает транслятору, что требуется создать бинарные файлы в формате ELF. Следует отметить, что формат elf64 позволяет создавать исполняемый код, работающий под 64-битными версиями Linux. Для 32-битных версий ОС указываем в качестве формата просто elf.
- NASM всегда создаёт выходные файлы в текущем каталоге



3) Расширенный синтаксис командной строки

NASM 4.3.3.

- Полный вариант командной строки nasm выглядит следующим образом:

```
nasm [-@ косвенный_файл_настроек] [-o объектный_файл] [-f  
  ↪ формат_объектного_файла] [-l листинг] [параметры...] [--]  
  ↪ исходный_файл
```

- Выполним следующую команду:

```
masidorenko@masidorenko:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst hello.asm  
hello.asm:1: warning: label alone on a line without a colon might be in error [-w+label-orphan]  
masidorenko@masidorenko:~/work/arch-pc/lab04$
```

- 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 были созданы

```
masidorenko@masidorenko:~/work/arch-pc/lab04$ ls  
hello.asm  hello.o  list.lst  obj.o  
masidorenko@masidorenko:~/work/arch-pc/lab04$
```

- Для более подробной информации см. man nasm. Для получения

списка форматов объектного файла см. `nasm -hf`.

```
masidorenko@masidorenko: ~/work/arch-pc/lab04
masidorenko@masidorenko:~/work/arch-pc/lab04$ man nasm
```

```
NASM(1)                                The Netwide Assembler Project                                NASM(1)
NAME
    nasm - the Netwide Assembler, a portable 80x86 assembler
SYNOPSIS
    nasm [-@ response file] [-f format] [-o outfile] [-l listfile] [options...] filename
DESCRIPTION
    The nasm command assembles the file filename and directs output to the file outfile if specified. If outfile
    is not specified, nasm will derive a default output file name from the name of its input file, usually by
    appending '.o' or '.obj', or by removing all extensions for a raw binary file. Failing that, the output file
    name will be 'nasm.out'.
OPTIONS
    -@ filename
        Causes nasm to process options from filename as if they were included on the command line.
    -a
        Causes nasm to assemble the given input file without first applying the macro preprocessor.
    -D[-d macro[=value]]
        Pre-defines a single-line macro.
    -E[-e]
        Causes nasm to preprocess the given input file, and write the output to stdout (or the specified output
        file name), and not actually assemble anything.
    -f format
        Specifies the output file format. To see a list of valid output formats, use the -hf option.
    -F format
        Specifies the debug information format. To see a list of valid output formats, use the -y option (for
        example -felf -y).
    -g
        Causes nasm to generate debug information.
    -gformat
        Equivalent to -g -F format.
    -h
        Causes nasm to exit immediately, after giving a summary of its invocation options.
Manual page nasm(1) line 1 (press h for help or q to quit)  Активация Windows
```

```
macro-params-single    single-line macro calls with wrong parameter count
negative-rep            regative %rep count [on]
number-overflow         numeric constant does not fit [on]
obsolete               all warnings prefixed with "obsolete-"
obsolete-nop           instruction obsolete and is a noop on the target
obsolete-removed       instruction obsolete and removed on the target CPU
obsolete-valid         instruction obsolete but valid on the target CPU
phase                 phase error during stabilization [off]
pragma                all warnings prefixed with "pragma-"
pragma-bad            malformed %pragma [off]
pragma-empty          empty %pragma directive [off]
pragma-na             %pragma not applicable to this compilation [off]
pragma-unknown        unknown %pragma facility or directive [off]
ptr                   non-NASM keyword used in other assemblers [on]
regsize               register size specification ignored [on]
unknown-warning       unknown warning in -W/-w or warning directive [off]
user                  %warning directives [on]
warn-stack-empty      warning stack empty [on]
zeroing               RESx in initialized section becomes zero [on]
zext-reloc            relocation zero-extended to match output format [on]
other                 any warning not specifiially mentioned above [on]

--limit-X val    set execution limit X
    passes        total number of passes [unlimited]
    stalled-passes number of passes without forward progress [1000]
    macro-levels   levels of macro expansion [10000]
    macro-tokens   tokens processed during single-line macro expansion [100000]
    mmacros        multi-line macros before final return [1000000]
    rep            %rep count [1000000]
    eval          expression evaluation descent [8192]
    lines         total source lines processed [2000000000]
masidorenko@masidorenko:~/work/arch-pc/lab04$
```

4) Компоновщик LD 4.4.



- Как видно из схемы на рис, чтобы получить исполняемую программу, объектный файл необходимо передать на обработку компоновщику:

```
masidorenko@masidorenko:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
```

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

```
masidorenko@masidorenko:~/work/arch-pc/lab04$ ls  
hello hello.asm hello.o list.lst obj.o
```

- Компоновщик ld не предполагает по умолчанию расширений для файлов, но принято использовать следующие расширения:
 - `o` – для объектных файлов;
 - без расширения – для исполняемых файлов;
 - `map` – для файлов схемы программы;
 - `lib` – для библиотек.
- Ключ `-o` с последующим значением задаёт в данном случае имя создаваемого исполняемого файла.
- Выполните следующую команду:

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

- (Вопрос: Какое имя будет иметь исполняемый файл?
Ответ: main)

- (Вопрос: Какое имя имеет объектный файл из которого собран этот исполняемый файл? Ответ: obj.o)
- Формат командной строки LD можно увидеть, набрав `ld --help`. Для получения более подробной информации см. `man ld`.

```
masidorenko@masidorenko:~/work/arch-pc/lab04$ ld --help
Использование ld [параметры] файл...
Параметры:
  -a КЛЮЧЕВОЕ СЛОВО          Управление общей библиотекой для совместимости с HP/UX
  -A АРХИТЕКТУРА, --architecture АРХИТЕКТУРА      Задать архитектуру
  -b ЦЕЛЬ, --format ЦЕЛЬ      Задать цель для следующих входных файлов
  -c ФАЙЛ, --mri-script ФАЙЛ  Прочитать сценарий компоновщика в формате MRI
  -d, --dc, --dp              Принудительно делать общие символы определенными
  --dependency-file ФАЙЛ      Write dependency file
  --force-group-allocation    Принудительно удалить членов группы из групп
  -e АДРЕС, --entry АДРЕС     Задать начальный адрес
  -E, --export-dynamic         Экспортировать все динамические символы
  --no-export-dynamic          Отменить действие --export-dynamic
  --enable-non-contiguous-regions
                                Enable support of non-contiguous memory regions
  --enable-non-contiguous-regions-warnings
                                Enable warnings when --enable-non-contiguous-regions may cause unexpected behavior
  -EB                          Компоновать объекты с прямым порядком байтов
  -EL                          Компоновать объекты с обратным порядком байтов
  -f SHLIB, --auxiliary SHLIB  Вспомогательный фильтр таблицы символов общих объектов
  -F SHLIB, --filter SHLIB     Фильтр для таблицы символов общих объектов
  -g                           Игнорируется
  -G РАЗМЕР, --gpsize РАЗМЕР   Размер маленьких данных (если не указан, то берётся из --shared)
  -h ИМЯ_ФАЙЛА, --soname ИМЯ_ФАЙЛА
                                Задать внутреннее имя общей библиотеки
  -I ПРОГРАММА, --dynamic-linker ПРОГРАММА          Назначить ПРОГРАММУ в качестве используемого динамического компоновщика
  --no-dynamic-linker          Создать исполняемый файл без заголовка программного интерпретатора
  -l LIBNAME, --library LIBNAME
                                Искать библиотеку с именем LIBNAME
  -L КАТАЛОГ, --library-path КАТАЛОГ               Добавить КАТАЛОГ к пути поиска библиотек
```

```
masidorenko@masidorenko: ~
masidorenko@masidorenko:~$ man ld
```

```
SYNOPSIS
ld [options] objfile ...

DESCRIPTION
ld combines a number of object and archive files, relocates their data and ties up symbol references. Usually the last step in compiling a program is to run ld.

ld accepts Linker Command Language files written in a superset of AT&T's Link Editor Command Language syntax, to provide explicit and total control over the linking process.

This man page does not describe the command language; see the ld entry in "info" for full details on the command language and on other aspects of the GNU linker.

This version of ld uses the general purpose BFD libraries to operate on object files. This allows ld to read, combine, and write object files in many different formats---for example, COFF or "a.out". Different formats may be linked together to produce any available kind of object file.

Aside from its flexibility, the GNU linker is more helpful than other linkers in providing diagnostic information. Many linkers abandon execution immediately upon encountering an error; whenever possible, ld continues executing, allowing you to identify other errors (or, in some cases, to get an output file in spite of the error).

The GNU linker ld is meant to cover a broad range of situations, and to be as compatible as possible with other linkers. As a result, you have many choices to control its behavior.

OPTIONS
The linker supports a plethora of command-line options, but in actual practice few of them are used in any particular context. For instance, a frequent use of ld is to link standard Unix object files on a standard, supported Unix system. On such a system, to link a file "hello.o":

ld -o <output> /lib/crt0.o hello.o -lc

This tells ld to produce a file called output as the result of linking the file "/lib/crt0.o" with "hello.o" and the library "libc.a", which will come from the standard search directories. (See the discussion of the -l option below.)

Some of the command-line options to ld may be specified at any point in the command line. However, options which refer to files, such as -l or -L, cause the file to be read at the point at which the option appears in the command line, relative to the object files and other file options. Repeating non-file options with a
```

5) Запуск исполняемого файла 4.4.1.

- Запустить на выполнение созданный исполняемый файл, находящийся в текущем каталоге, можно, набрав в командной строке:


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

6) Самостоятельная работа 4.5.


Ход работы:

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

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

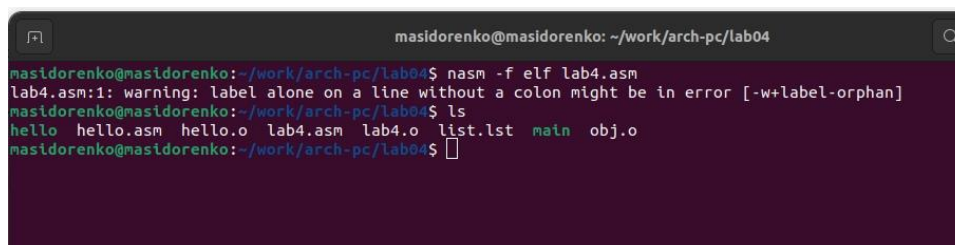
2. С помощью любого текстового редактора внесём изменения в текст программы в файле lab4.asm так, чтобы вместо Hello world! на экран выводилась строка с моей фамилией и именем.

```
masidorenko@masidorenko:~/work/arch-pc/lab04$ gedit lab4.asm
```



```
1 hello.asm
2 SECTION .data          ; Начало секции данных
3   hello: DB 'Сидоренко Максим!',10 ; 'Сидоренко Максим' плюс
4   ; символ перевода строки
5   helloLen: EQU $-hello ; Длина строки hello
6 SECTION .text          ; Начало секции кода
7   GLOBAL _start
8
9 _start:                ; Точка входа в программу
10  mov eax,4             ; Системный вызов для записи (sys_write)
11  mov ebx,1             ; Описатель файла '1' - стандартный вывод
12  mov ecx,hello          ; Адрес строки hello в ecx
13  mov edx,helloLen       ; Размер строки hello
14  int 80h               ; Вызов ядра
15
16  mov eax,1             ; Системный вызов для выхода (sys_exit)
17  mov ebx,0             ; Выход с кодом возврата '0' (без ошибок)
18  int 80h               ; Вызов ядра
```

3. Оттранслируем полученный текст программы lab4.asm в объектный файл. Выполним компоновку объектного файла и запустим получившийся исполняемый файл.



```
masidorenko@masidorenko:~/work/arch-pc/lab04$ nasm -f elf lab4.asm
lab4.asm:1: warning: label alone on a line without a colon might be in error [-w+label-orphan]
masidorenko@masidorenko:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab4.asm  lab4.o  list.lst  main  obj.o
masidorenko@masidorenko:~/work/arch-pc/lab04$ ./lab4
```

```
masidorenko@masidorenko:~/work/arch-pc/lab04$ nasm -o lab4obj.o -f elf -g -l lab4list.lst lab4.asm
lab4.asm:1: warning: label alone on a line without a colon might be in error [-w+label-orphan]
masidorenko@masidorenko:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o lab4 lab4.asm lab4list.lst lab4.o lab4obj.o list.lst main obj.o
masidorenko@masidorenko:~/work/arch-pc/lab04$

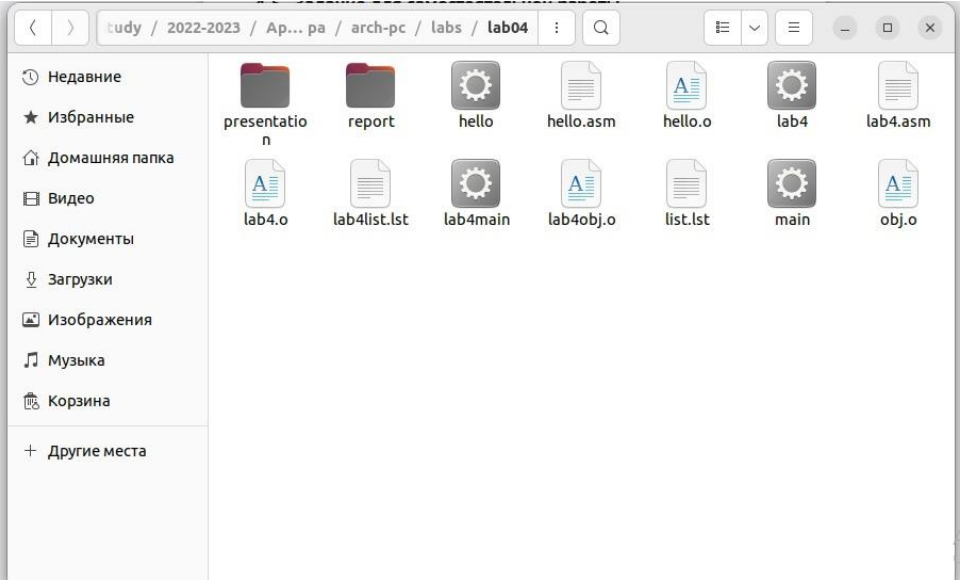
masidorenko@masidorenko:~/work/arch-pc/lab04$ ld -m elf_i386 lab4.o -o lab4
masidorenko@masidorenko:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o lab4 lab4.asm lab4list.lst lab4.o lab4obj.o list.lst main obj.o
masidorenko@masidorenko:~/work/arch-pc/lab04$

masidorenko@masidorenko:~/work/arch-pc/lab04$ ld -m elf_i386 lab4obj.o -o lab4main
masidorenko@masidorenko:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o lab4 lab4.asm lab4list.lst lab4main lab4.o lab4obj.o list.lst main obj.o
masidorenko@masidorenko:~/work/arch-pc/lab04$

masidorenko@masidorenko:~/work/arch-pc/lab04$ ./lab4
Сидоренко Максим
masidorenko@masidorenko:~/work/arch-pc/lab04$
```

4. Скопируем файлы hello.asm и lab4.asm в наш локальный репозиторий в каталог ~/work/study/2022-2023/"Архитектура компьютера"/archpc/labs/lab04/. Загрузим файлы на Github.

```
masidorenko@masidorenko:~/work/arch-pc/lab04$ cp hello hello.asm hello.o lab4 lab4.asm lab4.o lab4list.lst lab4main lab4obj.o list.lst main obj.o
~/work/study/2022-2023/"Архитектура компьютера"/arch-pc/labs/lab04/
```



```
masidorenko@masidorenko: ~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab04
masidorenko@masidorenko: ~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab04$ git add .
masidorenko@masidorenko: ~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab04$ git commit -an 'feat(main): add files lab-4'
[master c87fa3f] feat(main): add files lab-4
12 files changed, 76 insertions(+)
create mode 100755 labs/lab04/hello
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/hello.o
create mode 100755 labs/lab04/lab4
create mode 100644 labs/lab04/lab4.asm
create mode 100644 labs/lab04/lab4.o
create mode 100644 labs/lab04/lab4list.lst
create mode 100755 labs/lab04/lab4main
create mode 100644 labs/lab04/lab4obj.o
create mode 100644 labs/lab04/list.lst
create mode 100755 labs/lab04/main
create mode 100644 labs/lab04/obj.o
masidorenko@masidorenko: ~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab04$ git push
Перечисление объектов: 19, готово.
Подсчет объектов: 100% (19/19), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (19/19), готово.
Запись объектов: 100% (19/19), 3.98 KiB | 1.30 MiB/c, готово.
Всего 16 (изменений 10), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (10/10), completed with 2 local objects.
To github.com:MaximSidorenko/study_2022-2023_arh-pc.git
  1834987..c87fa3f  master -> master
masidorenko@masidorenko: ~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab04$
```

MaximSidorenko feat(main): add files lab-4		c87fa3f 21 seconds ago	History
..			
presentation	feat(main): make course structure	8 days ago	
report	feat(main): make course structure	8 days ago	
hello	feat(main): add files lab-4	21 seconds ago	
hello.asm	feat(main): add files lab-4	21 seconds ago	
hello.o	feat(main): add files lab-4	21 seconds ago	
lab4	feat(main): add files lab-4	21 seconds ago	
lab4.asm	feat(main): add files lab-4	21 seconds ago	
lab4.o	feat(main): add files lab-4	21 seconds ago	
lab4list.lst	feat(main): add files lab-4	21 seconds ago	
lab4main	feat(main): add files lab-4	21 seconds ago	
lab4obj.o	feat(main): add files lab-4	21 seconds ago	
list.lst	feat(main): add files lab-4	21 seconds ago	
main	feat(main): add files lab-4	21 seconds ago	
obj.o	feat(main): add files lab-4	21 seconds ago	

Вывод: При выполнении работы я освоил процедуры компиляции и сборки программ, написанных на ассемблере NASM

(ссылка на github)

(https://github.com/MaximSidorenko/study_2022-2023_arh-pc)

