Архитектура компьютеров и операционные системы | Операционные системы

Лабораторная работа № 4. Создание и процесс обработки программ на языке ассемблера NASM

Акрур Имад НКАбд-06-24

Содержание

# 1 Цель работы

* Изучить основы работы с ассемблером NASM и научиться создавать, компилировать и запускать программы на языке ассемблера.

Вот ответы на вопросы самопроверки:

# 2 Вопросы для самопроверки

1. **Какие основные отличия ассемблерных программ от программ на языках высокого уровня?**
   * Ассемблерные программы ближе к машинному коду и требуют более детального управления оборудованием. В отличие от языков высокого уровня, таких как Python или Java, которые предоставляют абстракции и автоматизацию, ассемблерные программы требуют явного указания каждой команды и обращения к памяти.
2. **В чём состоит отличие инструкции от директивы на языке ассемблера?**
   * Инструкции являются командами, которые процессор выполняет (например, MOV, ADD), тогда как директивы не преобразуются в машинный код, а служат для управления компилятором (например, .data, .text).
3. **Перечислите основные правила оформления программ на языке ассемблера.**
   * Каждая команда должна располагаться на отдельной строке.
   * Синтаксис чувствителен к регистру, т.е. MOV и mov будут восприниматься как разные команды.
   * Метки должны начинаться с буквы, знака подчеркивания или точки.
   * Комментарии начинаются с ; и продолжаются до конца строки.
4. **Каковы этапы получения исполняемого файла?**
   * Набор текста программы в текстовом редакторе и сохранение в файл с расширением .asm.
   * Трансляция исходного файла в объектный код с помощью транслятора (например, NASM).
   * Компоновка объектного файла в исполняемый файл с помощью компоновщика (например, LD).
   * Запуск получившегося исполняемого файла.
5. **Каково назначение этапа трансляции?**
   * Этап трансляции преобразует текст программы, написанной на ассемблере, в объектный код, который может быть использован для создания исполняемого файла. На этом этапе проверяются синтаксические ошибки и создаются объектные файлы.
6. **Каково назначение этапа компоновки?**
   * Этап компоновки объединяет один или несколько объектных файлов, а также библиотеки в единый исполняемый файл. Компоновщик разрешает внешние ссылки и размещает код в нужных адресах памяти.
7. **Какие файлы могут создаваться при трансляции программы, какие из них создаются по умолчанию?**
   * При трансляции могут создаваться объектные файлы (обычно с расширением .o), файлы листинга (с расширением .lst), а также файлы с отладочной информацией. По умолчанию создается только объектный файл.
8. **Каковы форматы файлов для NASM и LD?**
   * Для NASM: форматы могут включать elf, elf64, bin и другие, в зависимости от архитектуры и операционной системы.
   * Для LD: форматы включают elf\_i386, elf\_x86\_64 и другие.

# 3 Описание результатов выполнения лабораторной работы:

## 3.1 описание выполняемого задания:

### 3.1.1 Программа Hello world!

В этом разделе я рассмотрел пример простой программы на языке ассемблера NASM. Традиционно первая программа выводит приветственное сообщение “Hello world!” на экран.

1. Сначала я создал каталог для работы с программами на языке ассемблера NASM:

* mkdir -p ~/work/arch-pc/lab04



Рис. 1: рисунок 1

1. Затем я перешёл в созданный каталог:

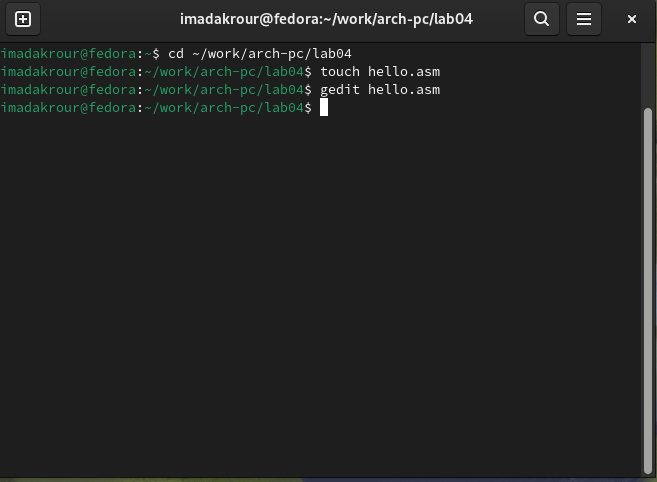
* cd ~/work/arch-pc/lab04

1. Я создал текстовый файл с именем hello.asm:

* touch hello.asm

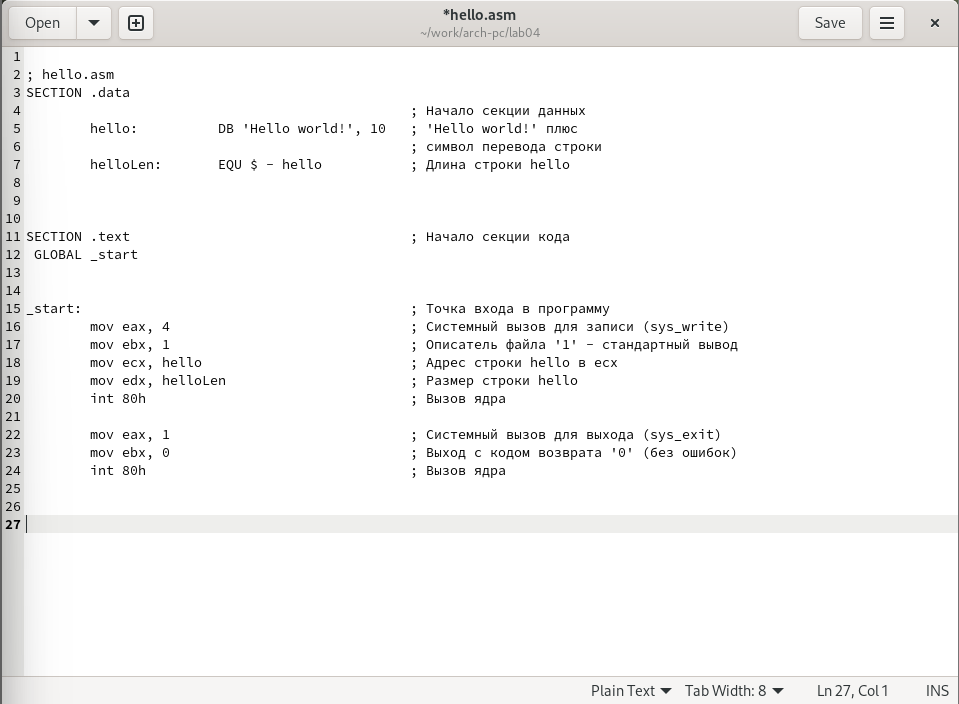
1. После этого я открыл этот файл с помощью текстового редактора gedit:

* gedit hello.asm

* 
* Рис. 2: рисунок 2

1. Я ввёл в него следующий текст:

* ; hello.asm  
  SECTION .data  
  hello:  
  ; Начало секции данных  
  DB 'Hello world!', 10 ; 'Hello world!' плюс  
  ; символ перевода строки  
  helloLen: EQU $ - hello  
  ; Длина строки hello  
  SECTION .text  
  GLOBAL \_start  
  \_start:  
  mov eax, 4  
  mov ebx, 1  
  mov ecx, hello  
  ; Начало секции кода  
  ; Точка входа в программу  
  ; Системный вызов для записи (sys\_write)  
  ; Описатель файла '1' - стандартный вывод  
  ; Адрес строки hello в ecx  
  mov edx, helloLen ; Размер строки hello  
  int 80h  
  ; Вызов ядра  
  mov eax, 1  
  mov ebx, 0  
  int 80h  
  ; Системный вызов для выхода (sys\_exit)  
  ; Выход с кодом возврата '0' (без ошибок)  
  ; Вызов ядра

* 
* Рис. 3: рисунок 3

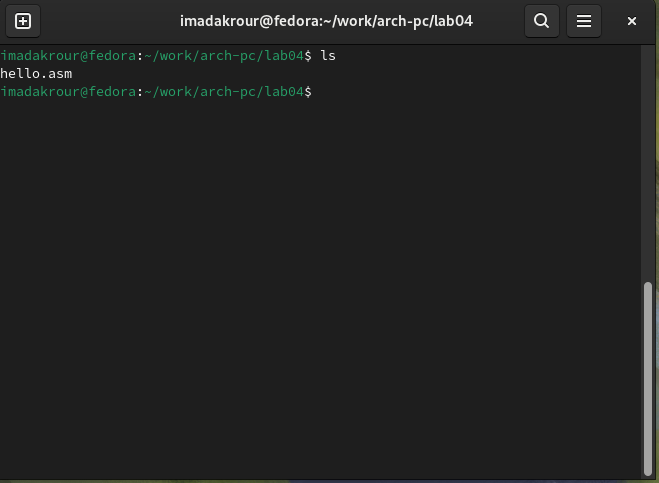


Рис. 4: рисунок 4

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.

### 3.1.2 Транслятор NASM

Я использовал NASM для превращения текста программы в объектный код. Например, для компиляции приведённого выше текста программы «Hello World» я написал:

nasm -f elf hello.asm

Если текст программы был набран без ошибок, то транслятор преобразовал текст программы из файла hello.asm в объектный код, который записался в файл hello.o. Таким образом, имена всех файлов получились из имени входного файла и расширения по умолчанию. При наличии ошибок объектный файл не создаётся, а после запуска транслятора появляются сообщения об ошибках или предупреждения.

Я проверил, что объектный файл был создан, используя команду ls:

ls

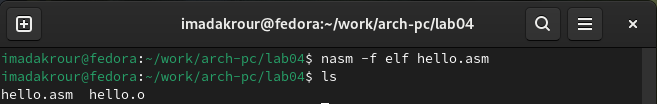


Рис. 5: рисунок 5

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 я указал в качестве формата просто elf. NASM всегда создаёт выходные файлы в текущем каталоге.

### 3.1.3 Расширенный синтаксис командной строки NASM

Я выполнил следующую команду:

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

Эта команда скомпилировала исходный файл hello.asm в obj.o (опция -o позволяет задать имя объектного файла, в данном случае obj.o), при этом формат выходного файла был elf, и в него были включены символы для отладки (опция -g), кроме того, был создан файл листинга list.lst (опция -l).

Я проверил, что файлы были созданы, используя команду ls:

ls

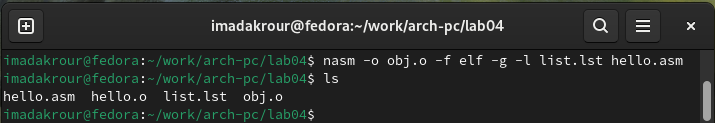


Рис. 6: рисунок 6

Для более подробной информации я обратился к man nasm. Для получения списка форматов объектного файла я использовал команду nasm -hf.

### 3.1.4 Компоновщик LD

Как видно из схемы на рис. 4.3, чтобы получить исполняемую программу, объектный файл необходимо передать на обработку компоновщику:

ld -m elf\_i386 hello.o -o hello

Я проверил, что исполняемый файл hello был создан, с помощью команды ls:

ls

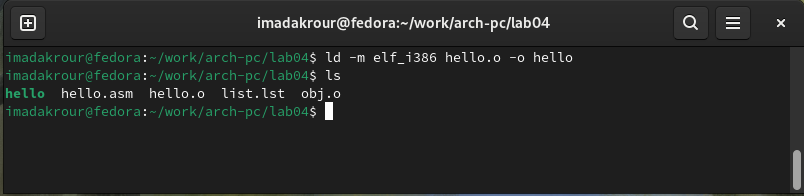


Рис. 7: рисунок 7

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.

Я выполнил следующую команду:

ld -m elf\_i386 obj.o -o main

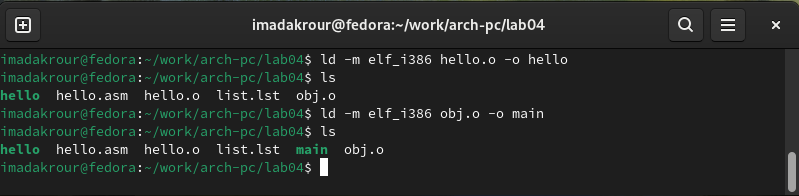


Рис. 8: рисунок 8

**Исполняемый файл будет называться main, а объектный файл, из которого был собран этот исполняемый файл, называется obj.o.**

### 3.1.5 Запуск исполняемого файла

Чтобы запустить созданный исполняемый файл, находящийся в текущем каталоге, я набрал в командной строке:

./hello

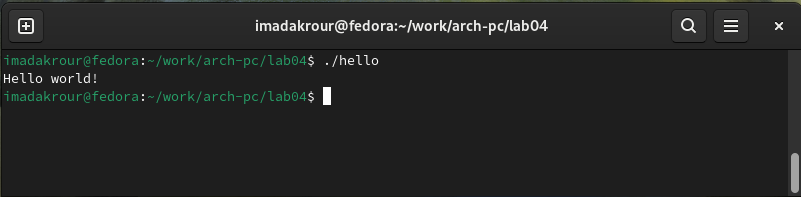


Рис. 9: рисунок 9

Эта команда указывает системе выполнить файл hello, который является исполняемым. Точка и косая черта (./) указывают на то, что файл находится в текущем каталоге.

## 3.2 выводы по результатам выполнения заданий :

В результате выполнения лабораторной работы я изучил основные этапы работы с ассемблером NASM и научился создавать, компилировать и запускать программы на языке ассемблера. Работа с NASM позволяет лучше понять архитектуру компьютера и внутренние механизмы работы операционной системы.

# 4 Описание результатов выполнения заданий для самостоятельной работы:

## 4.1 описание выполняемого задания;

### 4.1.1 Создание копии файла

В каталоге ~/work/arch-pc/lab04 я создал копию файла hello.asm с именем lab4.asm, используя команду cp:

cp hello.asm lab4.asm

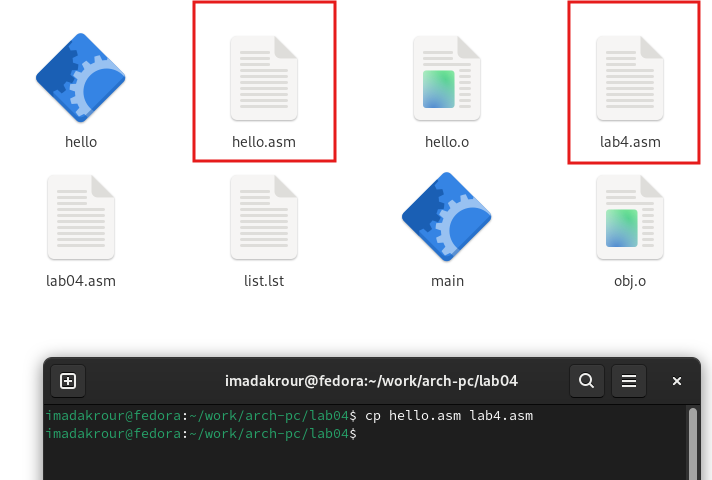


Рис. 10: рисунок 10

Эта команда создаёт точную копию исходного файла hello.asm, переименовывая его в lab4.asm. Это удобно для внесения изменений в новую версию программы, сохраняя оригинал.

### 4.1.2 Внесение изменений в текст программы

Затем я открыл файл lab4.asm с помощью текстового редактора gedit:

gedit lab4.asm

В текст программы я внес изменения, чтобы вместо “Hello world!” на экран выводилась строка с моими фамилией и именем. Вот как я изменил содержимое файла:

; lab4.asm  
SECTION .data  
hello:  
DB 'Akrour Imad', 10 ; Изменено на 'Akrour Imad'  
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



Рис. 11: рисунок 11

Я заменил строку “Hello world!” на “Akrour Imad” , чтобы программа выводила моё имя на экран. Также я убедился, что длина строки обновлена соответствующим образом.

### 4.1.3 Трансляция и компоновка

Теперь я оттранслировал полученный текст программы lab4.asm в объектный файл, используя следующую команду:

nasm -f elf lab4.asm

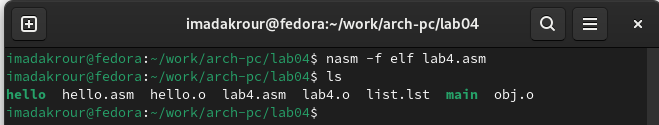


Рис. 12: рисунок 12

После успешной компиляции, я выполнил компоновку объектного файла, чтобы создать исполняемый файл, с помощью компоновщика ld:

ld -m elf\_i386 lab4.o -o lab4

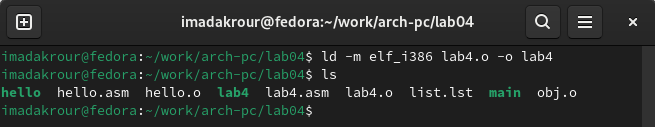


Рис. 13: рисунок 13

Теперь я запустил новую программу:

./lab4

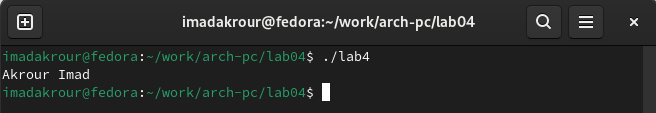


Рис. 14: рисунок 14

Если всё прошло успешно, программа должна была вывести “Иванов Иван” на экран, подтверждая правильность изменений, внесённых в файл.

### 4.1.4 Копирование файлов в локальный репозиторий

Для выполнения последнего шага я скопировал файлы hello.asm и lab4.asm в мой локальный репозиторий в каталог ~/work/study/2024-2025/"Архитектура компьютера"/arch-pc/labs/lab04/, используя команду:

cp ~/work/arch-pc/lab04/hello.asm ~/work/study/2024-2025/"Архитектура компьютера"/arch-pc/labs/lab04/  
cp ~/work/arch-pc/lab04/lab4.asm ~/work/study/2024-2025/"Архитектура компьютера"/arch-pc/labs/lab04/

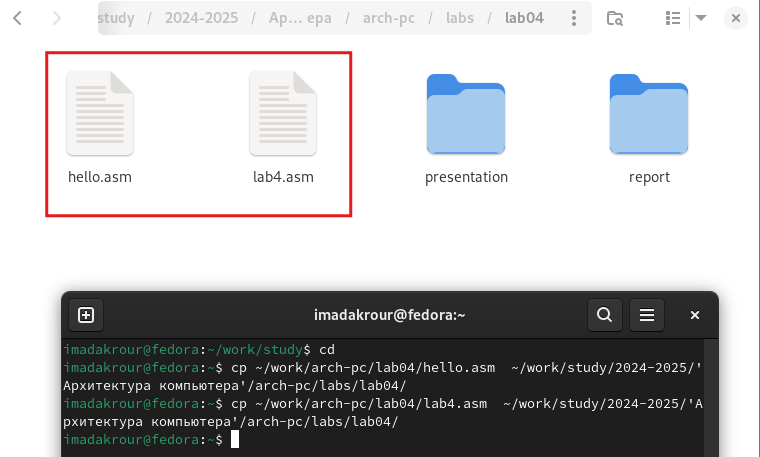


Рис. 15: рисунок 15

Эти команды копируют файлы в указанный каталог, что позволяет организовать и сохранить выполненные работы.

Затем я загрузил файлы на GitHub, следуя процессу, который обычно включает инициализацию репозитория (если это ещё не сделано), добавление файлов и коммит:

cd ~/work/study/2024-2025/"Архитектура компьютера"/arch-pc/labs/lab04/  
  
git add .  
git commit -m "Adding hello.asm & lab4.asm"  
git push

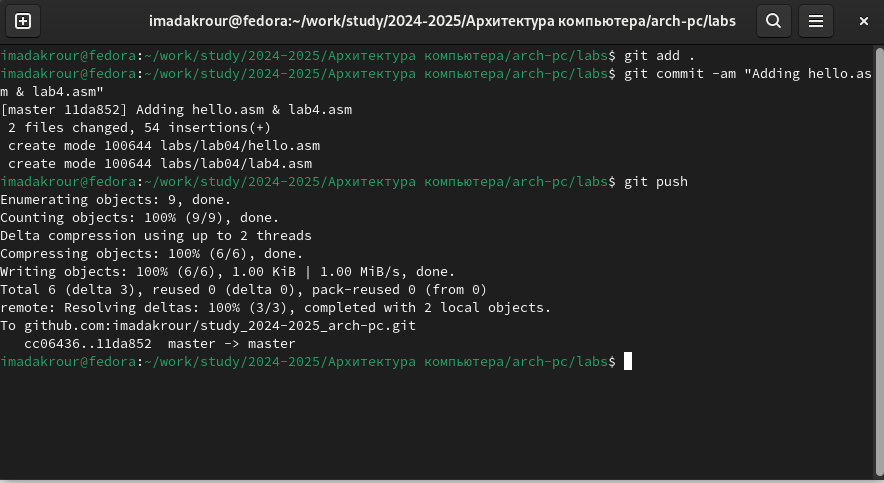


Рис. 16: рисунок 16

Я добавил изменения в локальный репозиторий и загрузил их на GitHub, чтобы сохранить и поделиться результатами работы.

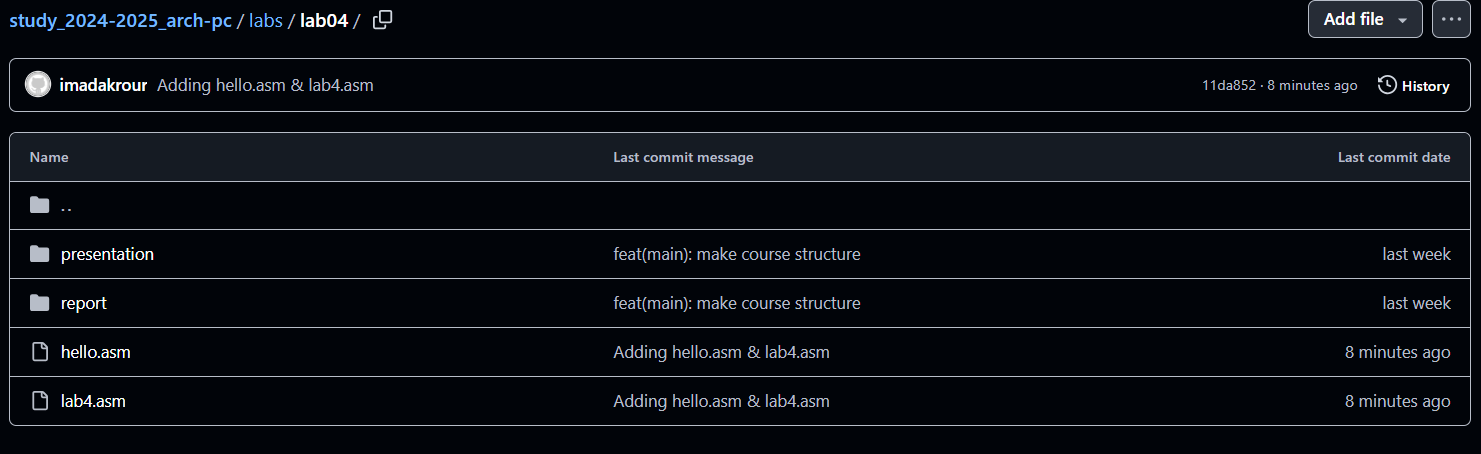


Рис. 17: рисунок 17

## 4.2 выводы по результатам выполнения заданий :

В результате выполнения задания для самостоятельной работы я научился создавать копии файлов, изменять программы на языке ассемблера, компилировать и запускать их, а также загружать результаты работы на GitHub. Это упражнение помогло мне лучше понять работу с NASM и методы организации проектов.

# 5 Выводы,согласованные с целью работы :

Лабораторная работа позволила мне освоить основные принципы программирования на ассемблере, от написания кода до его компиляции и запуска, что значительно углубило мои знания в низкоуровневом программировании.