Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Толстых Александра Андреевна

Содержание

# 1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Задание

1. Реализация подпрограмм в NASM.
2. Отладка программам с помощью GDB.
3. Задание для самостоятельной работы.

# 3 Выполнение лабораторной работы

## 3.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создю файл lab09-1.asm (рис. 1).



Рис. 1: Создание каталога и файла

Ввожу в файл lab09-1.asm текст программы из листинга 9.1 (рис. 2).

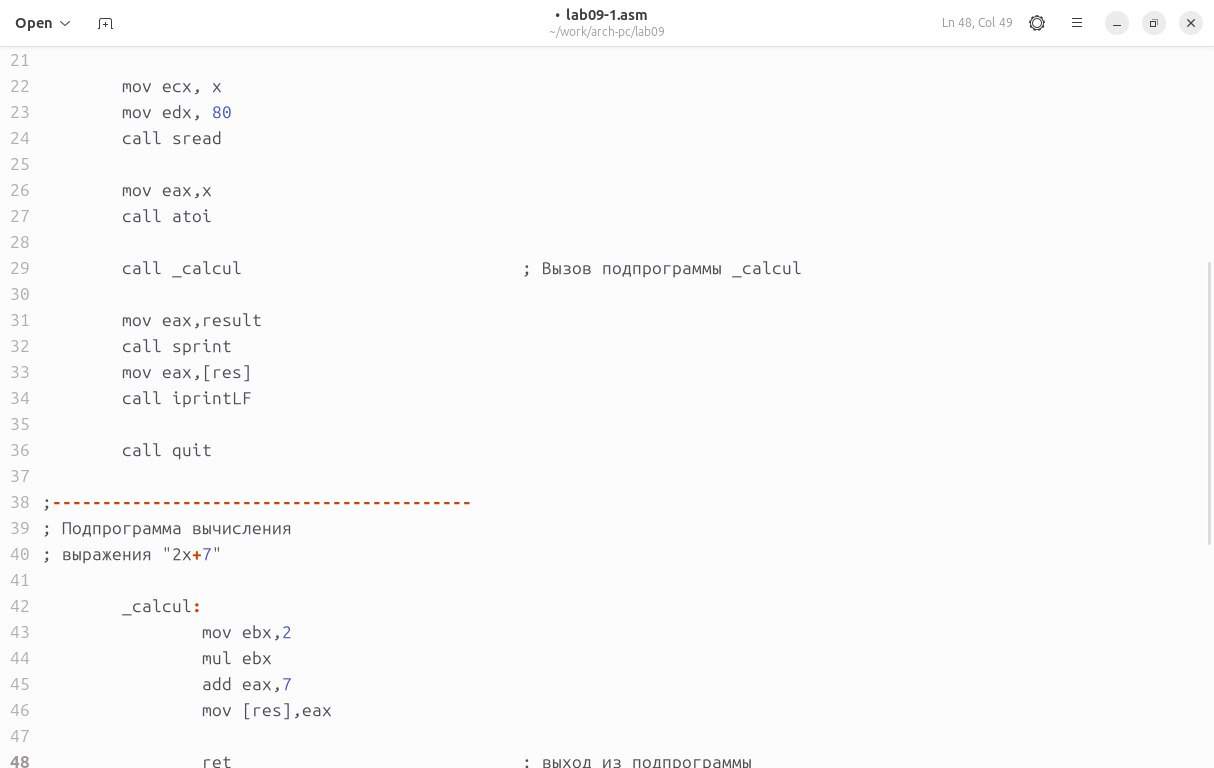


Рис. 2: Написание программы

Создаю исполняемый файл и проверяю его работу (рис. 3).

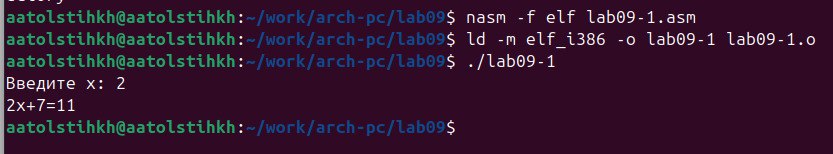


Рис. 3: Запуск программы

Изменяю текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x) = 2x + 7, g(x) = 3x − 1 (рис. 4).

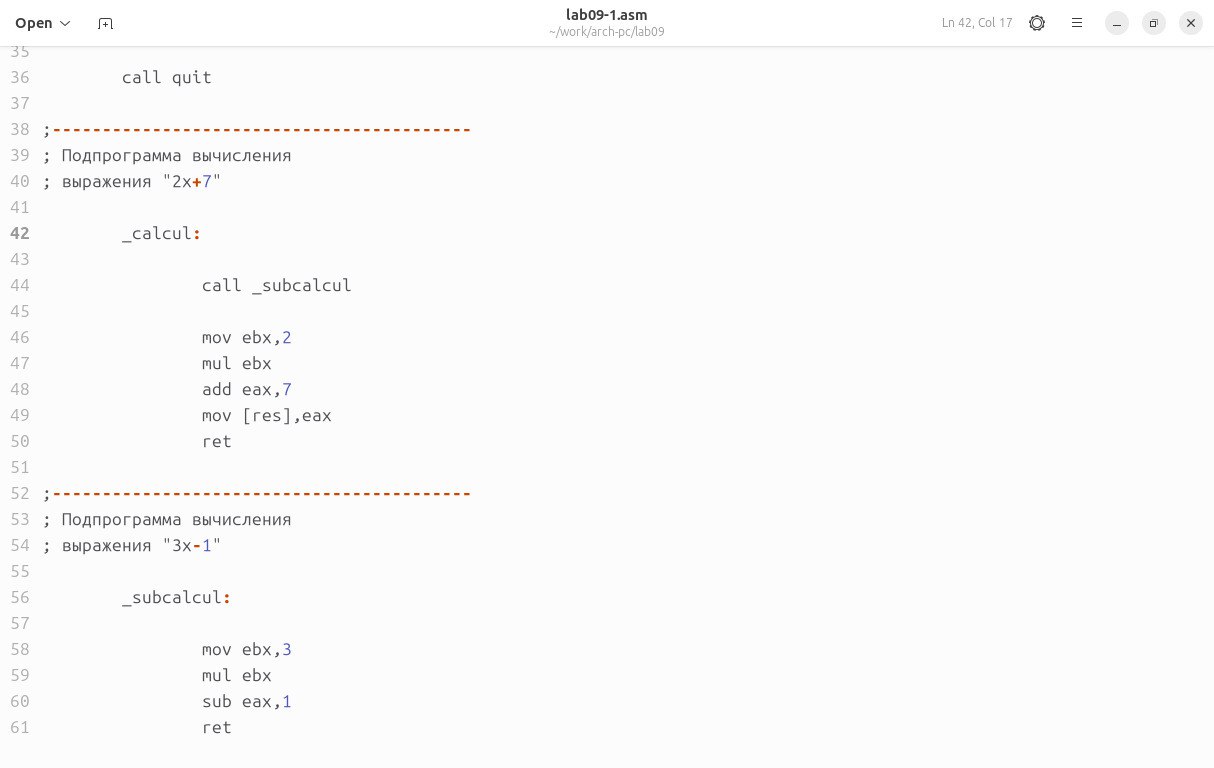


Рис. 4: Изменение программы

Создаю исполняемый файл и проверяю его работу (рис. 5).

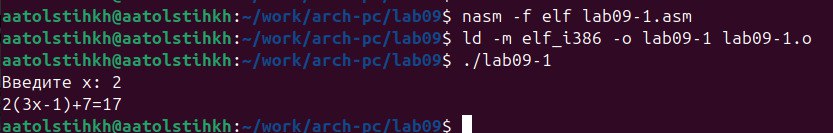


Рис. 5: Запуск программы

## 3.2 Отладка программам с помощью GDB

Создаю файл lab09-2.asm (рис. 6).

Рис. 6: Создание файла

Рис. 6: Создание файла

Ввожу в него текст из программы из Листинга 9.2 (рис. 7).

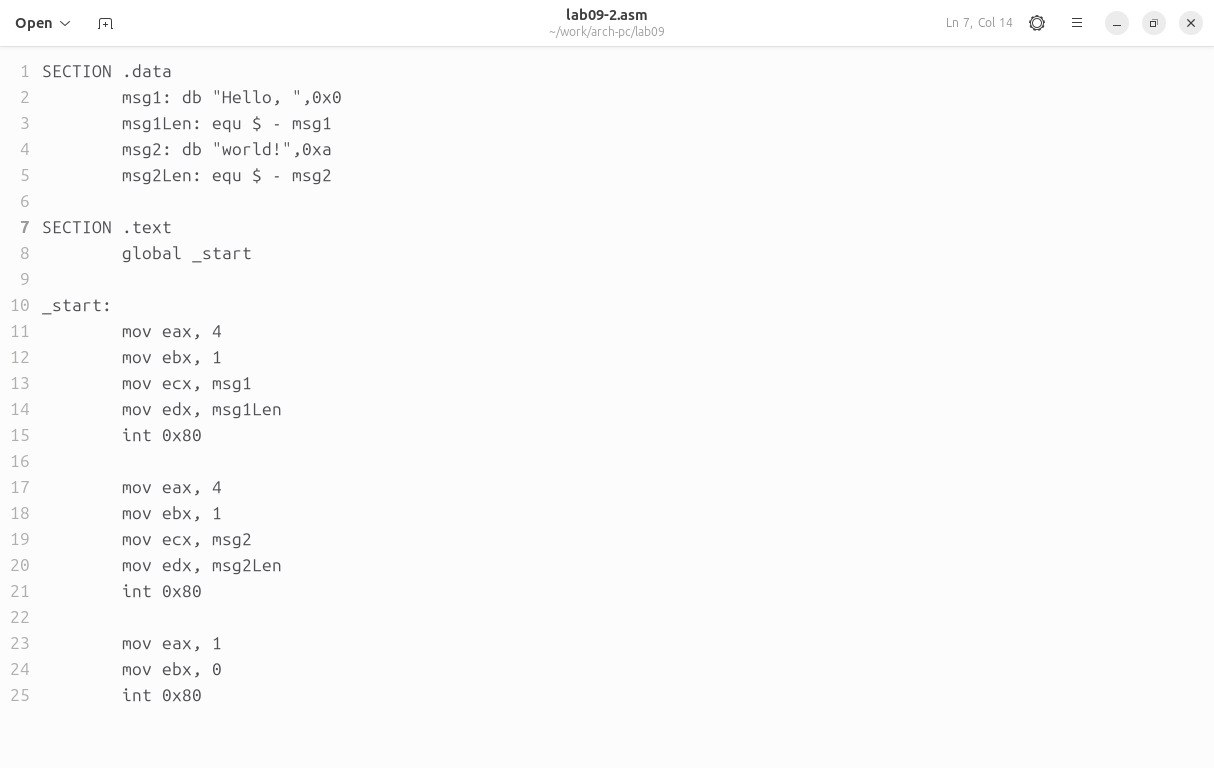


Рис. 7: Написание программы

Получаю исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’. Загружаю исполняемый файл в отладчик gdb (рис. 8).

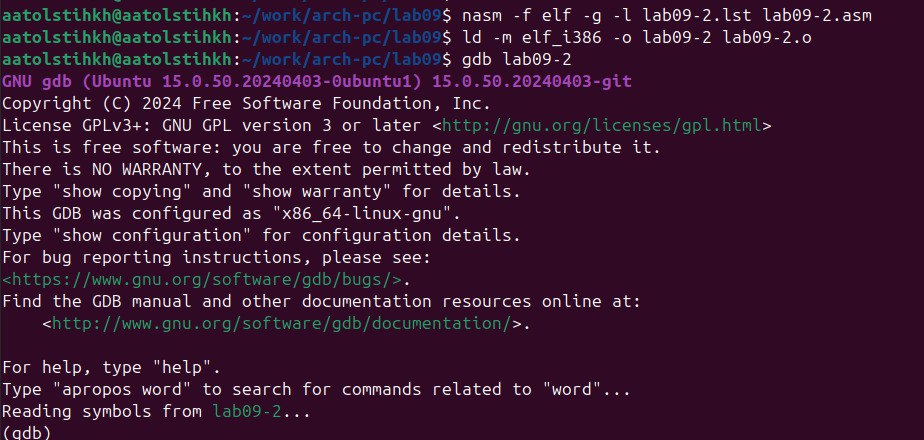


Рис. 8: Создание и загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 9).

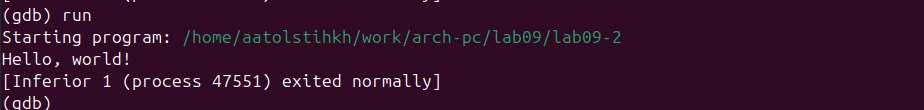


Рис. 9: Запуск программы

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаю её (рис. 10).

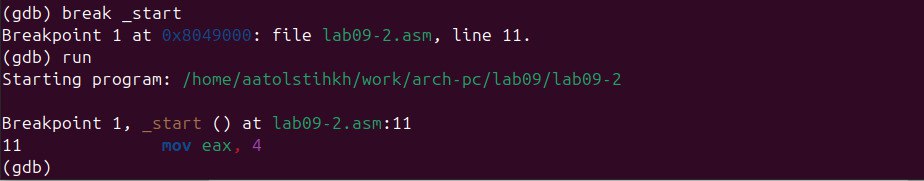


Рис. 10: Запуск программы

Смотрю дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. 11).

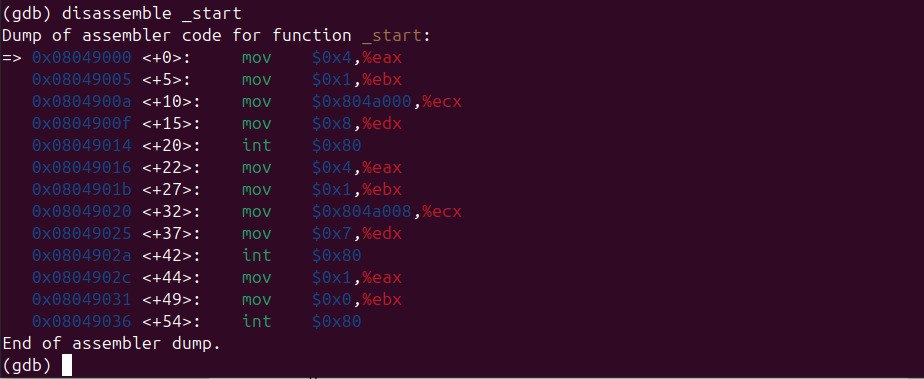


Рис. 11: Просмотр программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. 12). Отличие заключается в командах, в диссамилированном отображении в командах используют % и $, а в Intel отображение эти символы не используются. На такое отображение удобнее смотреть.

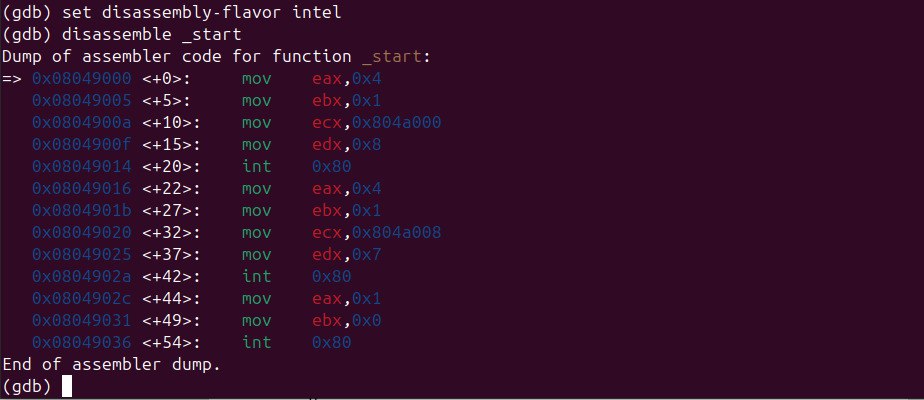


Рис. 12: Просмотр программы с другим отображением

Включаю режим псевдографики для более удобного анализа программы (рис. 13).

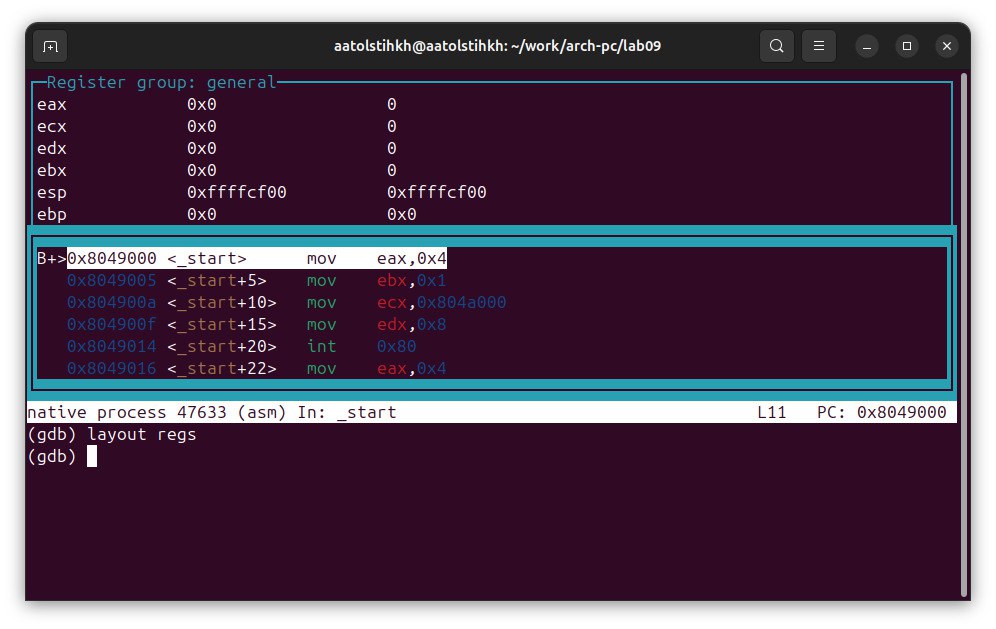


Рис. 13: Режим псевдографики

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяю это с помощью команды info breakpoints (рис. 14).

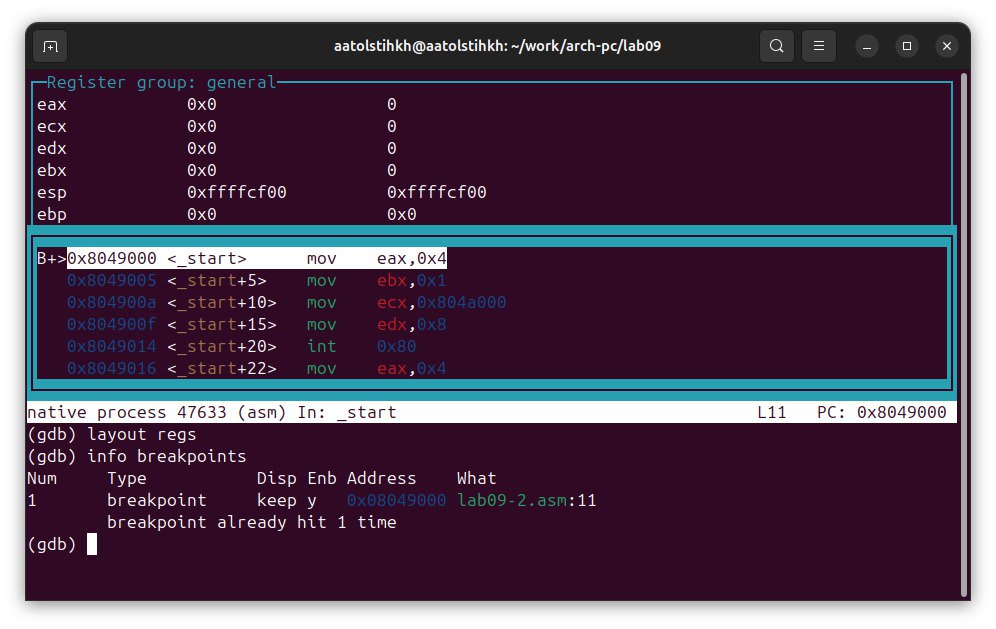


Рис. 14: Проверка точек останова

Устанавливаю еще одну точку останова по адресу предпоследней инструкции (mov ebx,0x0) и снова смотрю информацию о всех установленных точках останова (рис. 15).

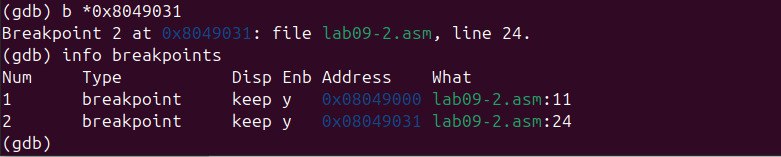


Рис. 15: Установка и просмотр точек останова

Выполняю 5 инструкций с помощью команды stepi (или si). Изменяются регистры ebx, ecx, edx, eax (рис. 16).



Рис. 16: Выполнение 5 инструкций

Смотрю значение переменной msg1 по имени (рис. 17).



Рис. 17: Просмотр значения переменной по имени

Смотрю значение переменной msg2 по адресу (рис. 18).



Рис. 18: Просмотр значения переменной по адресу

Изменяю первый символ переменной msg1 (рис. 19).

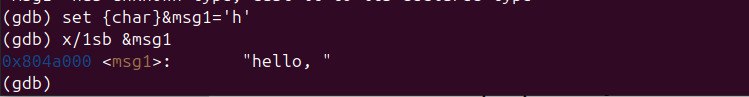


Рис. 19: Изменение переменной

Изменяю символ во второй переменной msg2 (рис. 20).

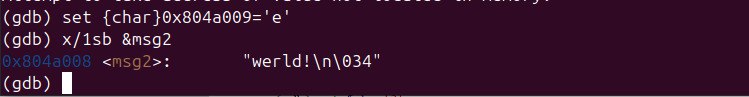


Рис. 20: Изменение переменной

Вывожу в различных форматах значение регистра edx (рис. 21).

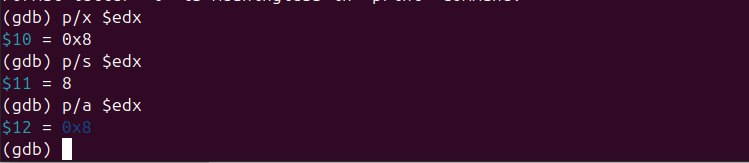


Рис. 21: Вывод значения регистра

С помощью команды set изменяю значение регистра ebx (рис. 22). Команда выводит два разных значения, так как в первый раз мы вносим значение 2, а во второй раз регистр равен двум.

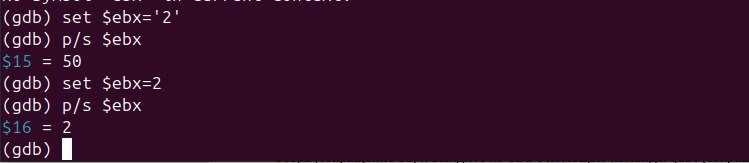


Рис. 22: Изменение значения регистра

Завершаю выполнение программы с помощью команды continue (сокращенно c) и выхожу из GDB с помощью команды quit (сокращенно q) (рис. 23).

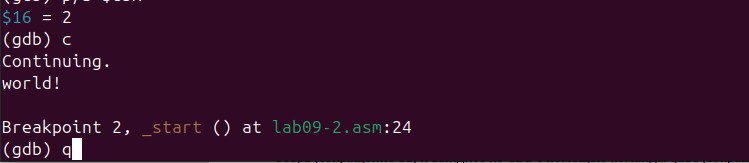


Рис. 23: Завершение выполнения и выход из отладчика

Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm (рис. 24).

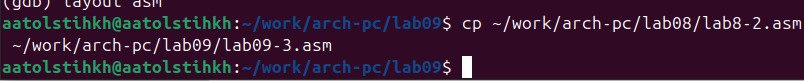


Рис. 24: Создание копии файла

Создаю исполняемый файл (рис. 25).

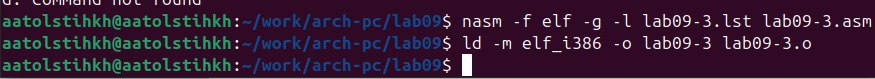


Рис. 25: Создание исполняемого файла

Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загружаю исполняемый файл в отладчик, указав аргументы (рис. 26).

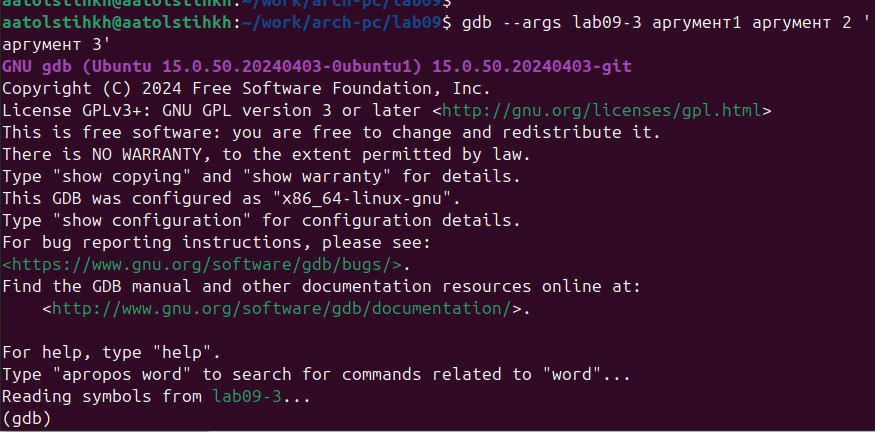


Рис. 26: Загрузка исполняемого файла в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее (рис. 27).

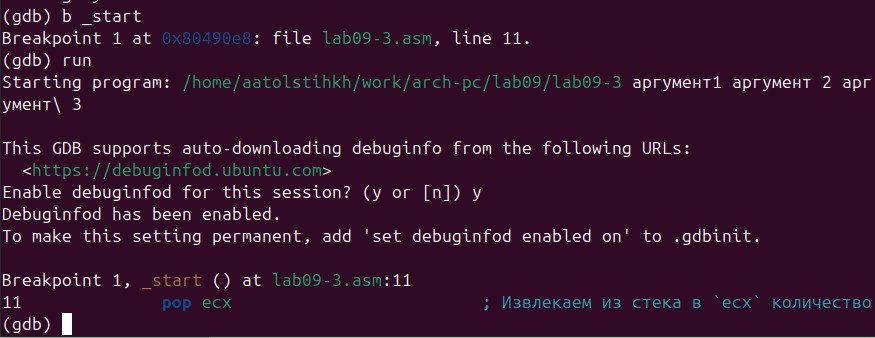


Рис. 27: Установка точки и запуск программы

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Убеждаюсь, что там число 5 (рис. 28).



Рис. 28: Проверка количества аргументов

Смотрю все позиции стека (рис. 29). По первому адресу хранится адрес, в остальных адресах хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт, и для того чтобы данные сохранялись нормально и без помех, компьютер использует новый стек для новой информации.

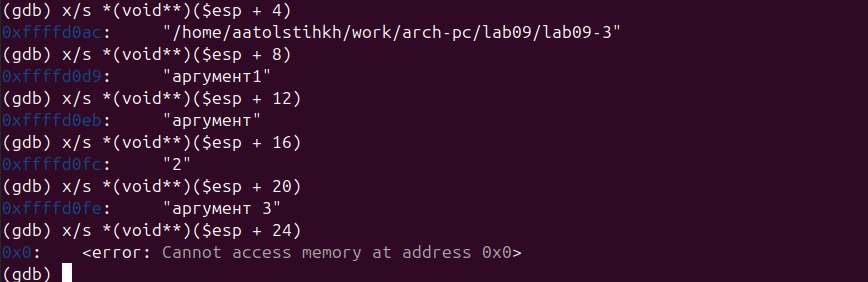


Рис. 29: Позиции стека

## 3.3 Задание для самостоятельной работы

Копирую файл из предыдущей лабораторной (рис. 30).

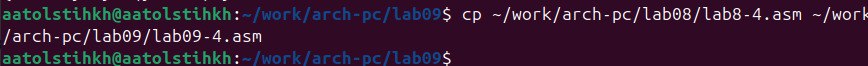


Рис. 30: Создание копии программы

Преобразую программу, реализовав вычисление значения функции f(x) как подпрограмму (рис. 31).

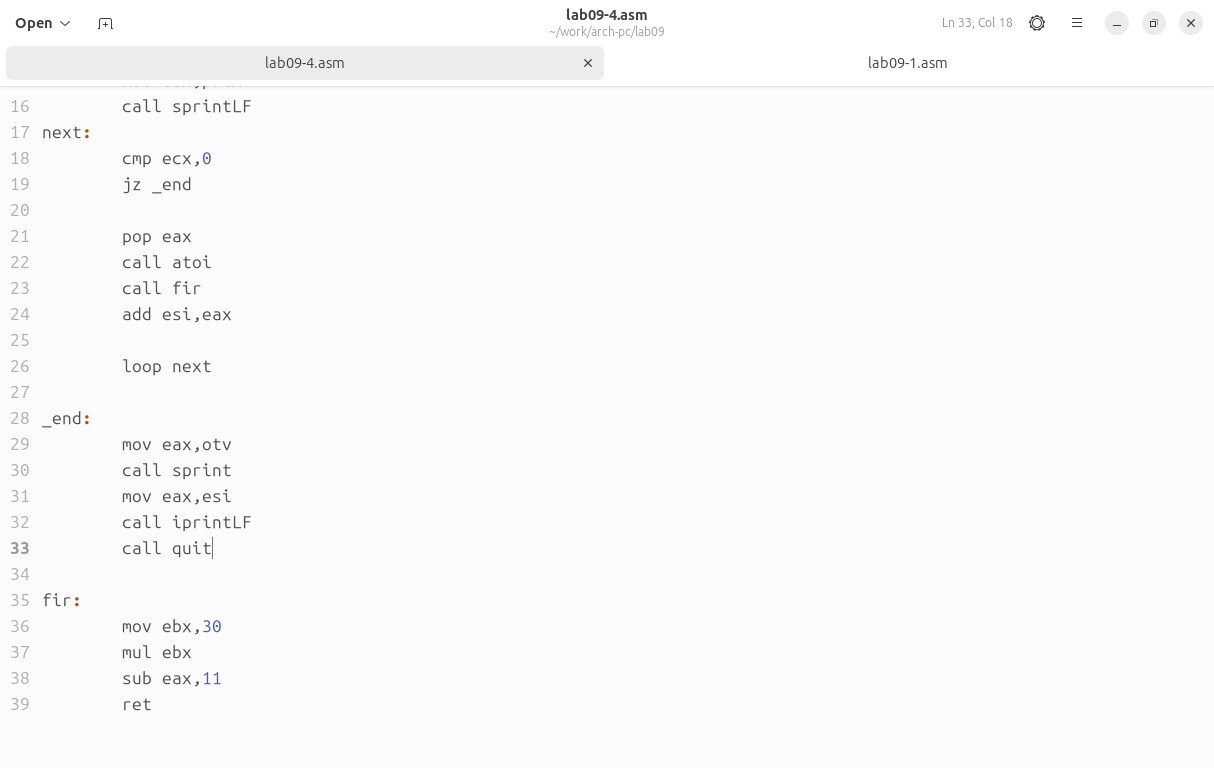


Рис. 31: Изменение программы

Создаю исполняемый файл и проверяю его работу (рис. 32).

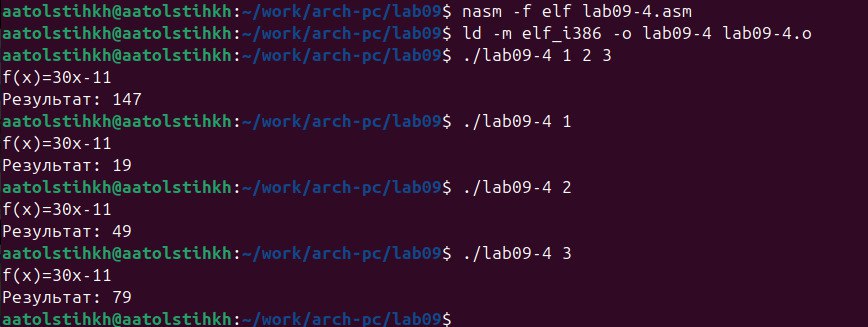


Рис. 32: Запуск программы

Создаю новый файл для дальнейшей работы с программой из листинга 9.3 (рис. 33).

Рис. 33: Создание файла

Рис. 33: Создание файла

Ввожу в него текст программы (рис. 34).

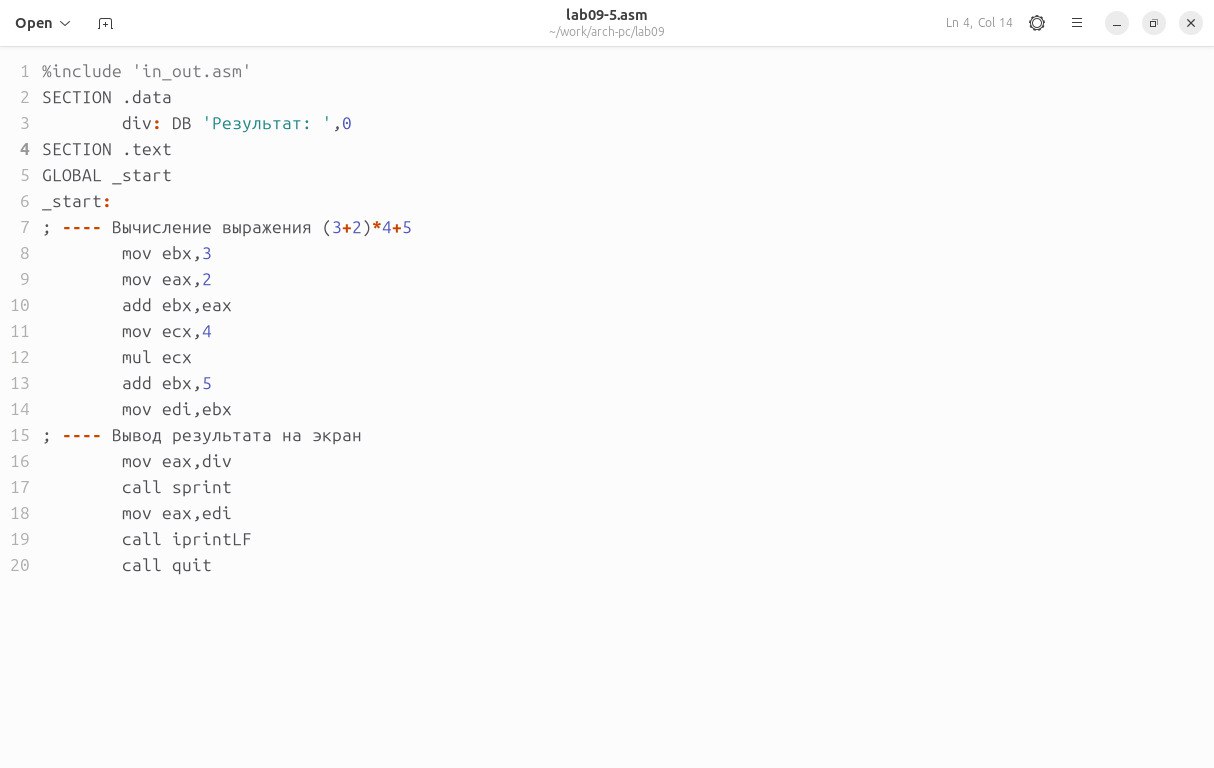


Рис. 34: Написание программы

Получаю исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’. Загружаю исполняемый файл в отладчик gdb и запускаю программу при помощи команды ‘r’ (рис. 35).

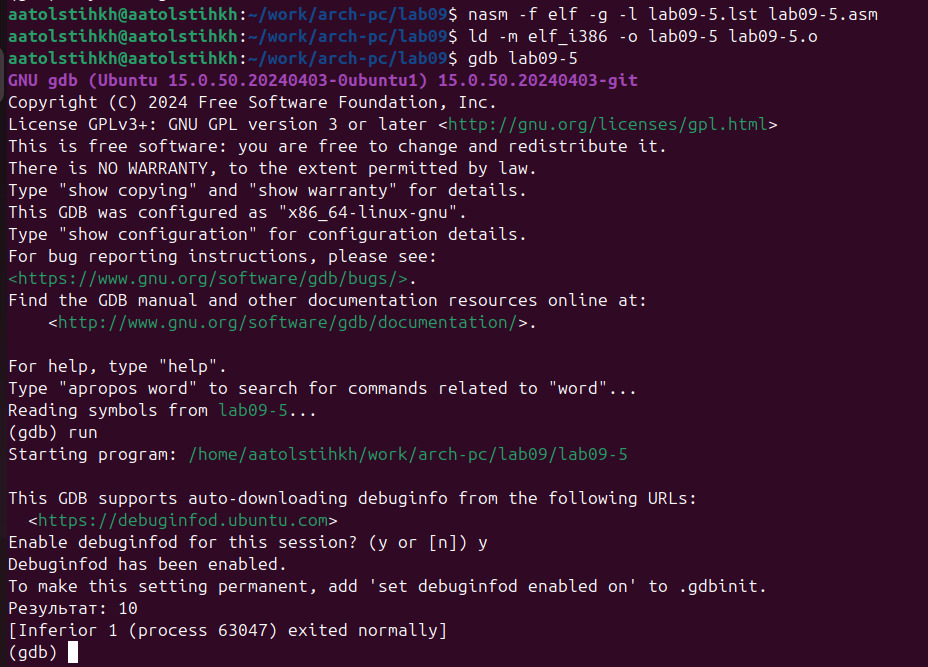


Рис. 35: Создание и загрузка исполняемого файла в отладчик

Результат программы неправильный, поэтому настраиваю отображение команд для дальнейшего изучения (рис. 36).

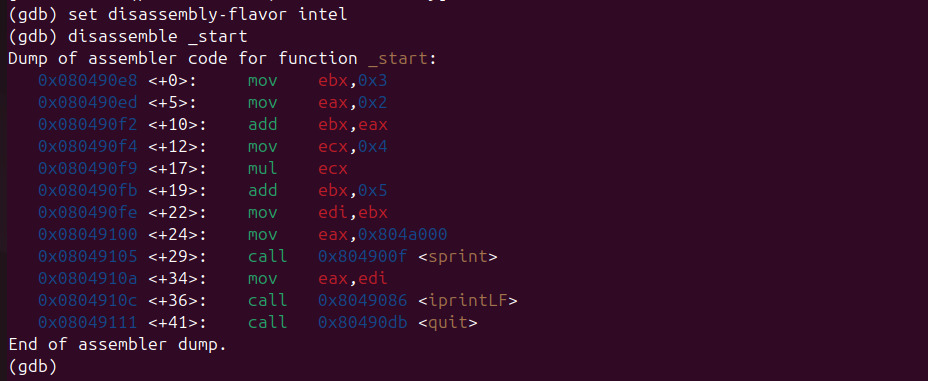


Рис. 36: Настройка отображения команд

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаю её (рис. 37).

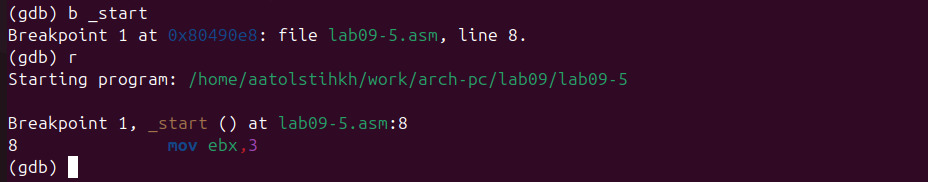


Рис. 37: Запуск программы

Анализируя регистры, замечаю, что некоторые стоят не на своих местах (рис. 38).

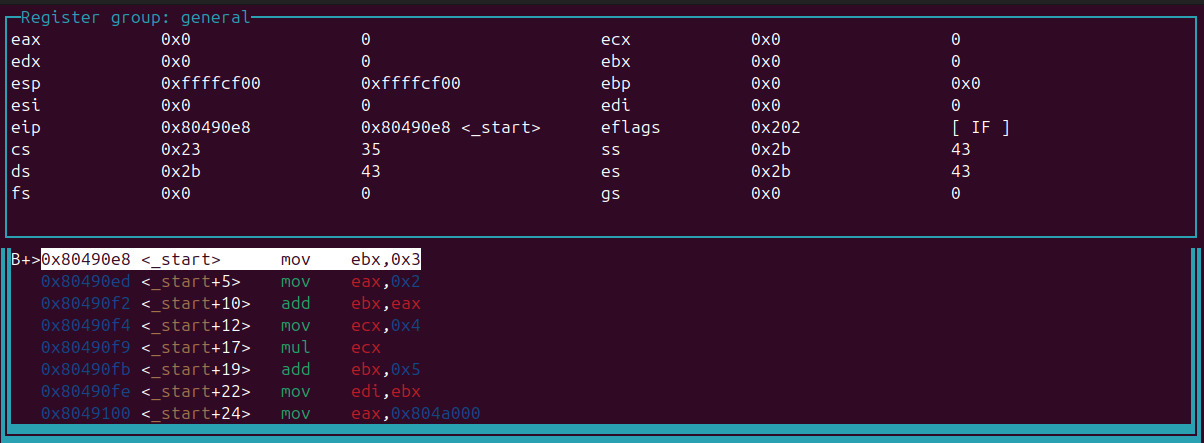


Рис. 38: Анализ регистров

Открываю программу и изменяю места регистров на правильные (рис. 39).

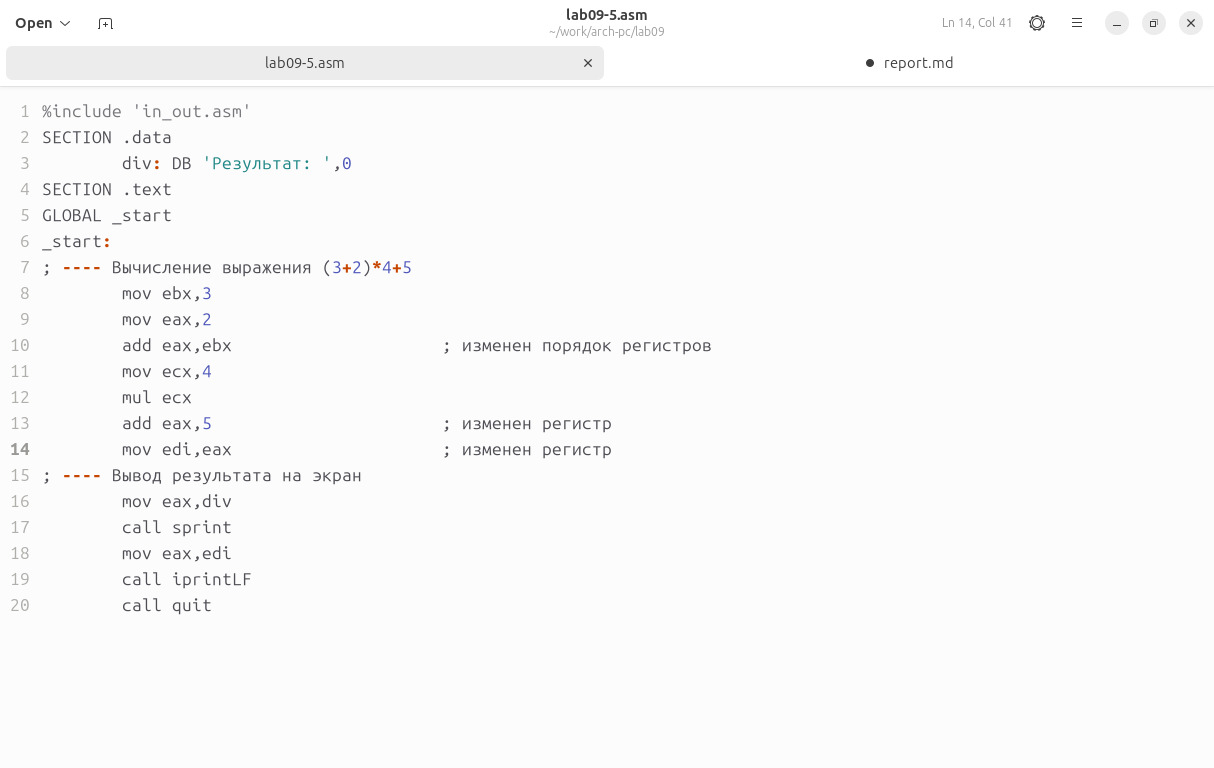


Рис. 39: Изменение регистров

После нахождения ошибки и изменения программы заново создаю исполняемый файл и запускаю его (рис. 40). Теперь программа работает корректно.

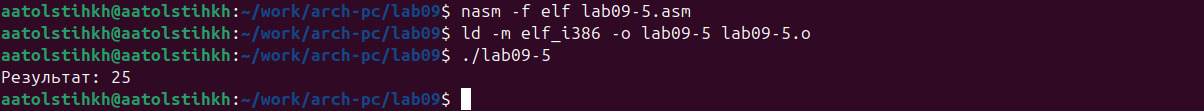


Рис. 40: Запуск программы

# 4 Выводы

В ходе выполнения лабораторной работы я приобрела навыки написания программ с использованием подпрограмм.А также познакомилась с методами отладки при помощи GDB и его основными возможностями.