Лабораторная работа №9.

Понятие подпрограммы. Отладчик GDB.

Жукова София Викторвна

Содержание

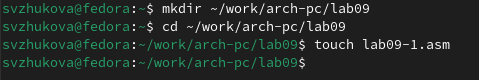
# Цель работы

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

# Выполнение лабораторной работы

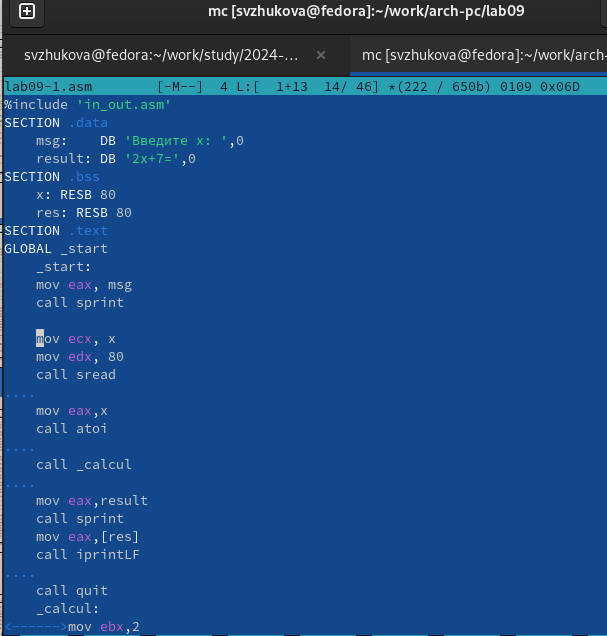
**1. Реализация подпрограмм в NASM**

Создадим каталог для выполнения лабораторной работы № 9, перейдем в него и со- здадим файл lab09-1.asm: (рис. [-@fig:001]).



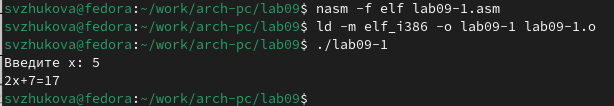
Создаем каталог и файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. [-@fig:002]).



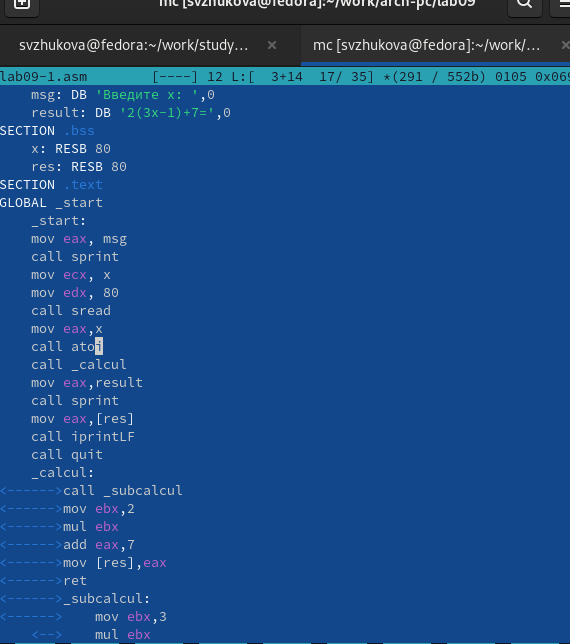
Заполняем файл

Создаем исполняемый файл и запускаем его (рис. [-@fig:003]).



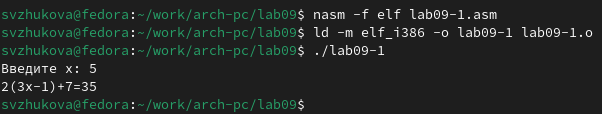
Запускаем файл и проверяем его работу

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму (рис. [-@fig:004]).



Изменяем файл

Создаем исполняемый файл и запускаем его (рис. @fig:005).



Запускаем файл и смотрим на его работу

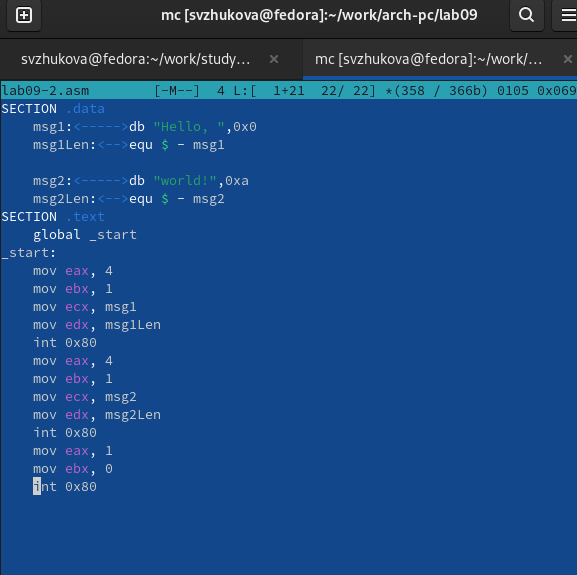
## Отладка программам с помощью GDB

Создаем новый файл в каталоге(рис. @fig:006).

Создаем файл

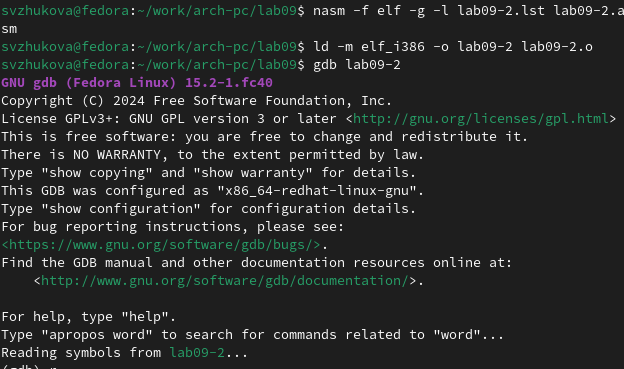
Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. @fig:007).



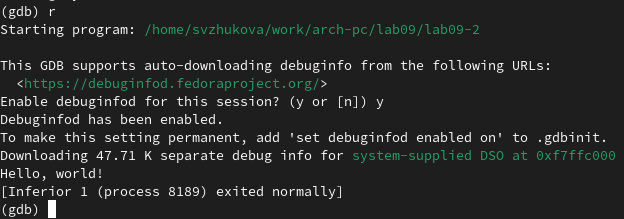
Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. @fig:008).



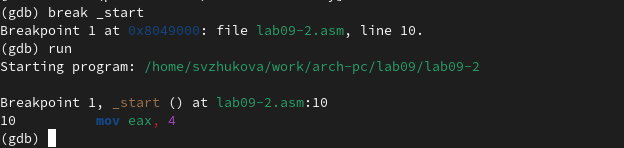
Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. @fig:009).



Запускаем программу командой run

Устанавливаем брейкпоинт на метку \_start и запускаем программу (рис. @fig:010).



Запускаем программу с брейкпоином

Смотрим дисассимилированный код программы с помощью команды disassemble, начиная с метки \_start(рис. @fig:011).



Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel’овским синтаксисом (рис. @fig:012).



Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах ATT и Intel:

1.Порядок операндов: В ATT синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный - вторым.

2.Разделители: В ATT синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

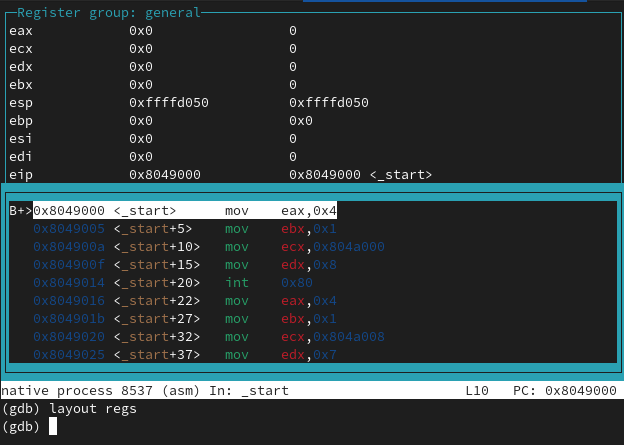
3.Префиксы размера операндов: В ATT синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (byte), “w” (word), “l” (long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”.

4.Знак операндов: В ATT синтаксисе операнды с позитивными значениями предваряются символом “$". В Intel синтаксисе операнды с позитивными значениями могут быть указаны без символа "$”.

5.Обозначение адресов: В ATT синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.

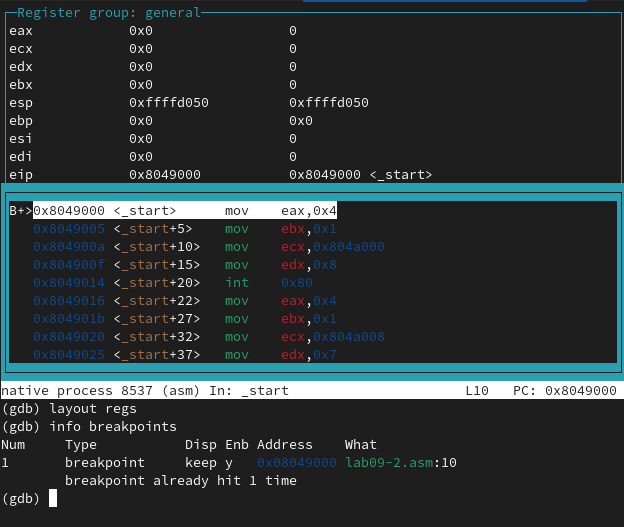
6.Обозначение регистров: В ATT синтаксисе обозначение регистра начинается с символа “%”. В Intel синтаксисе обозначение регистра может начинаться с символа “R” или “E” (например, “%eax” или “RAX”).

Включаем режим псевдографики (рис. @fig:013).



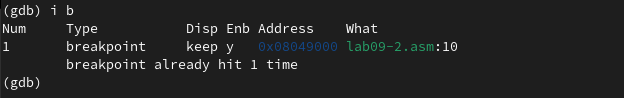
Включаем отображение регистров, их значений и результат дисассимилирования программы

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. @fig:014).



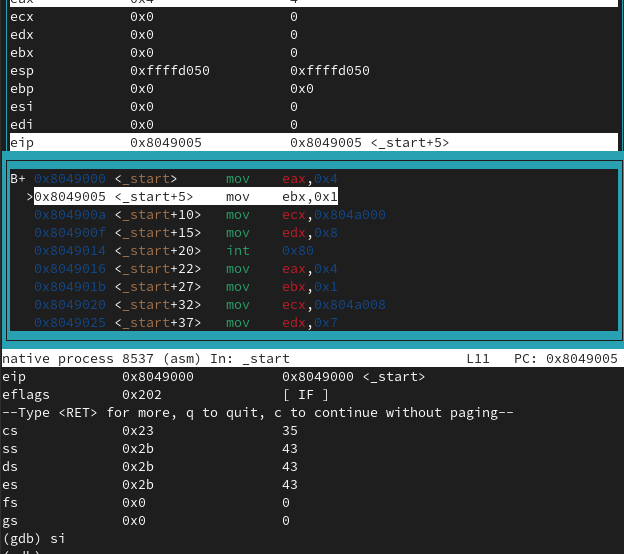
Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова (рис. @fig:015).



Смотрим информацию

Выполняем 5 инструкций командой si (рис. @fig:016).



Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx,eax, eip.

Смотрим значение переменной msg1 по имени (рис. @fig:017).

Смотрим значение переменной

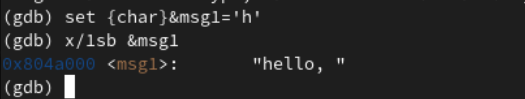
Смотрим значение переменной

Смотрим значение переменной msg2 по адресу (рис. @fig:018).

Смотрим значение переменной

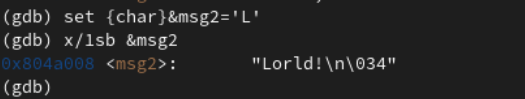
Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. @fig:019).



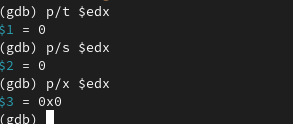
Меняем символ

Изменим первый символ переменной msg2 (рис. @fig:020).



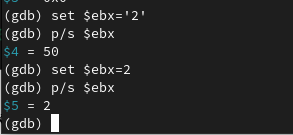
Меняем символ

Смотрим значение регистра edx в разных форматах (рис. @fig:021).



Смотрим значение регистра

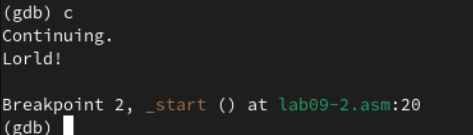
Изменяем регистор ebx (рис. @fig:022).



Изменяем регистор командой set

Выводится разные значения, так как команда без кеавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB (рис. @fig:023).



Прописываем команды c и quit

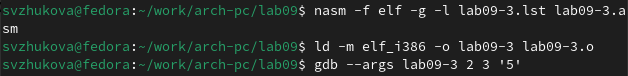
**3. Обработка аргументов командной строки в GDB**

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. @fig:024).

Копируем файл

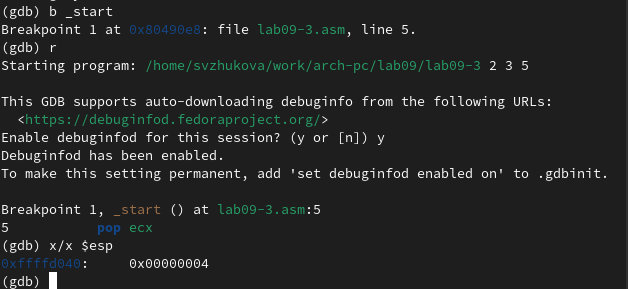
Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. @fig:025).



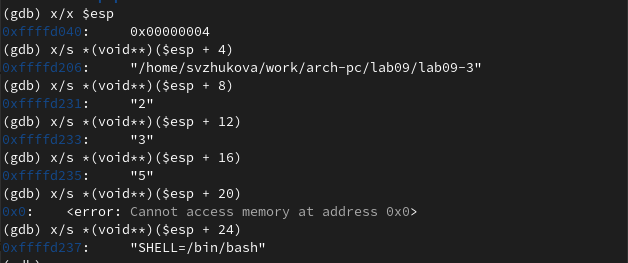
Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. @fig:026).



Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. @fig:027).



Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

##Задание для самостоятельной работы

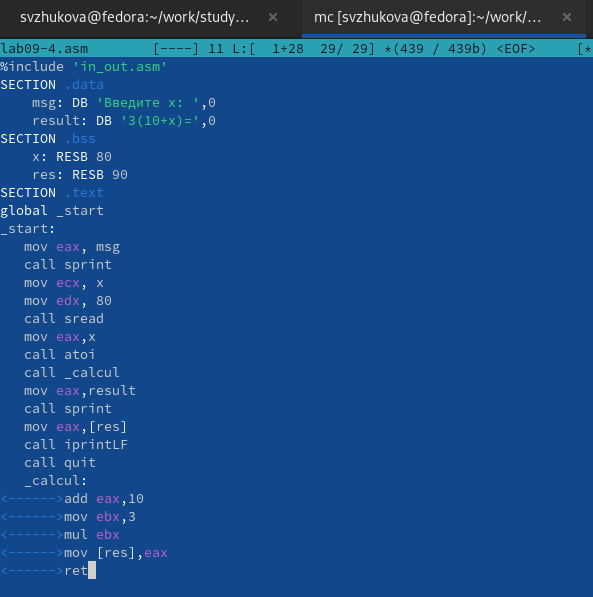
**Задание 1**

Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем lab09-3.asm (рис. @fig:028).

Копируем файл

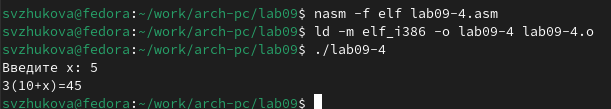
Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. @fig:029).



Изменяем файл

Создаем исполняемый файл и запускаем его (рис. @fig:030).



Проверяем работу программы

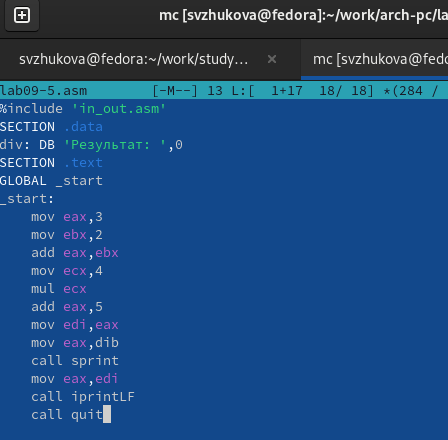
###Задание 2

Создаем новый файл в дирректории (рис. @fig:031).

Создаем файл

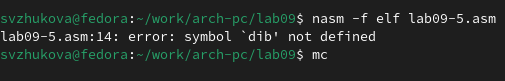
Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. @fig:032).



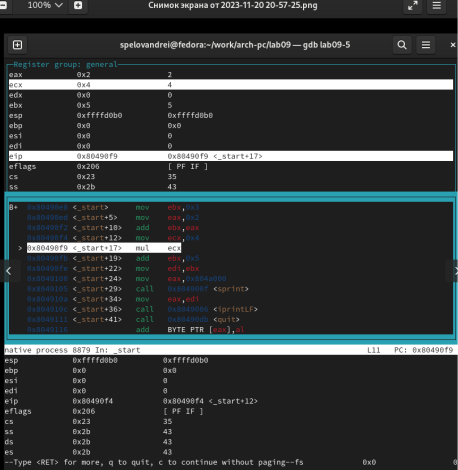
Изменяем файл

Создаем исполняемый файл и запускаем его (рис. @fig:033).



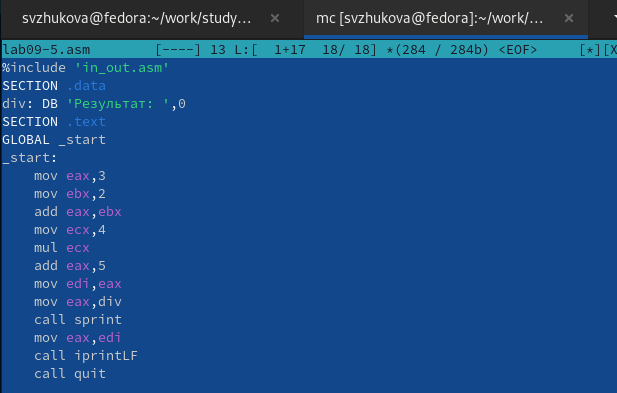
Создаем и смотрим на работу программы(работает неправильно)

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение решистров командой si (рис. @fig:034).



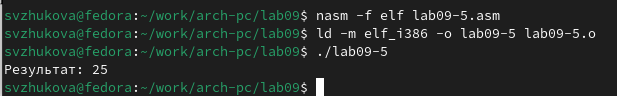
Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы (рис. @fig:035).



Меняем файл

Создаем исполняемый файл и запускаем его (рис. @fig:036).



Создаем и запускаем файл

# Выводы

Мы приобрели навыки написания программ с использованием подпрограмм. Познакомились с методами отладки при помощи GDB и его основными возможностями.