Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютера

Арфонос Дмитрий

Содержание

# 1 Цель работы

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

# 2 Выполнение лабораторной работы

**Шаг 1**

С помощью утилиты mkdir создаю директорию lab09, перехожу в нее и создаю файл для работы. (рис. [[1](#fig:001)])

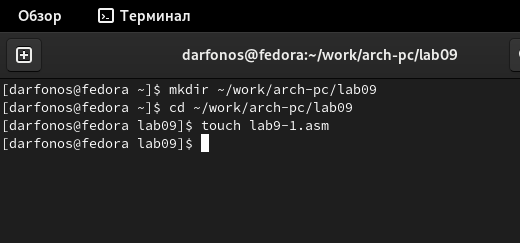


Figure 1: Создание директории

**Шаг 2**

Открываю созданный файл lab9-1.asm, вставляю в него программу с использованием подпрограммы(рис.[[2](#fig:002)]).

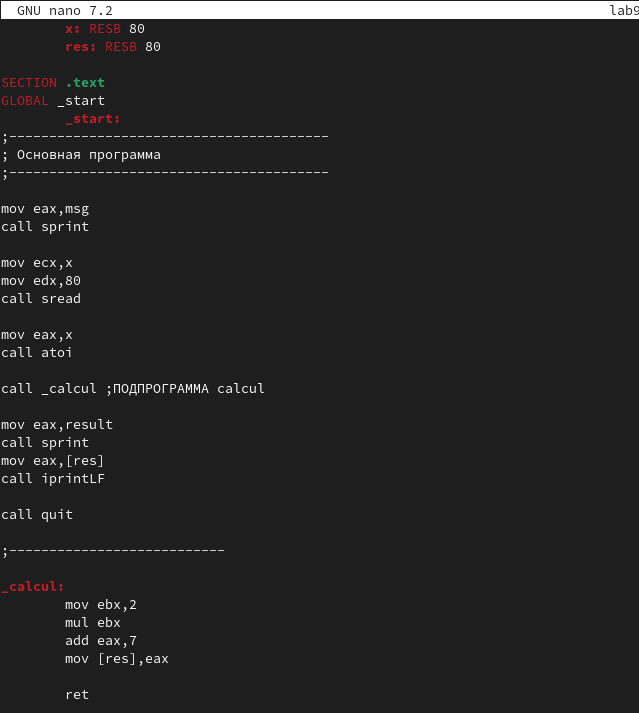


Figure 2: Редактирование файла

**Шаг 3**

Создаю исполняемый файл программы и запускаю его (рис. [[3](#fig:003)]).

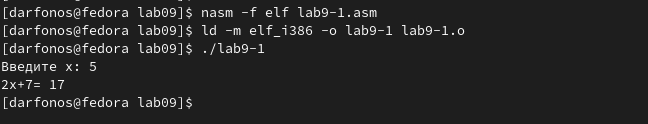


Figure 3: Запуск исполняемого файла

**Шаг 4**

Изменяю текст программы для вычисления композиции f от g, при g(x) = 3x-1. Создаю новую подпрограмму \_subcalcul для вычисления функции g (рис. [[4](#fig:004)]).

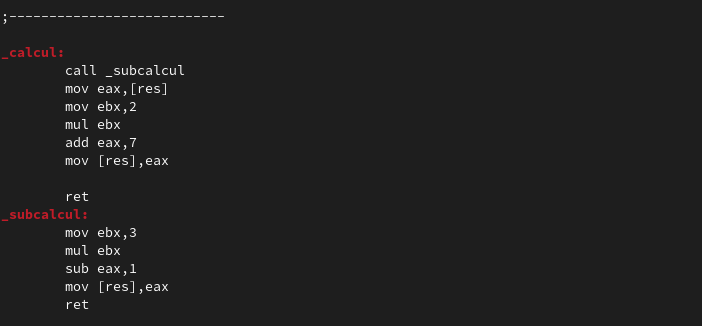


Figure 4: Редоктирование

**Шаг 5**

Создаю исполняемый файл и проверяю работу программы (рис. [[5](#fig:005)]).

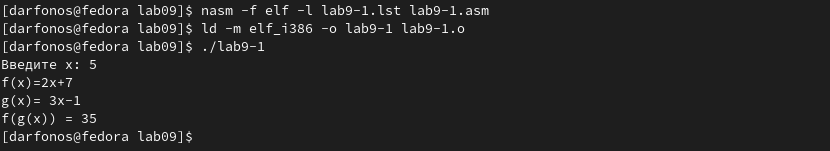


Figure 5: Запуск исполняемого файла

* Программа отработала верно!!

**Шаг 6**

Создаю новый файл lab9-2.asm и вставляю в него текст из Листинга 9.2 (рис. [[6](#fig:006)]).

 { #fig:006 width=80% }

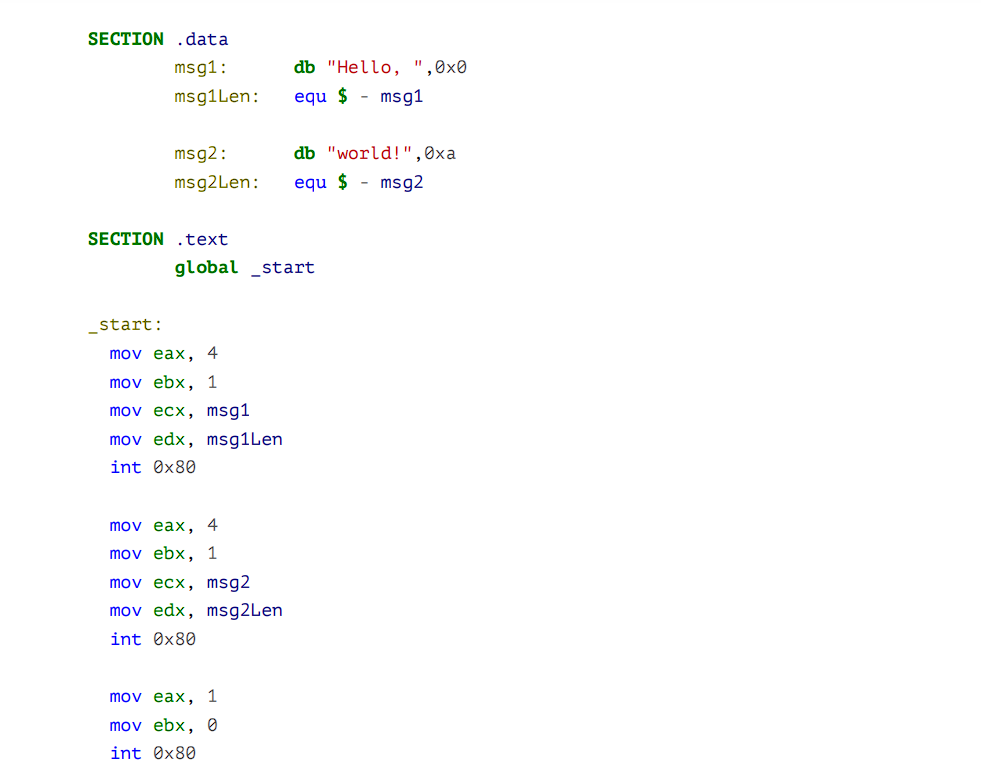


Figure 6: Редактирование программы

**Шаг 7**

Создаю исполняемый файл, файл листинга для работы с отладчиком GDB (рис. [[7](#fig:007)]).

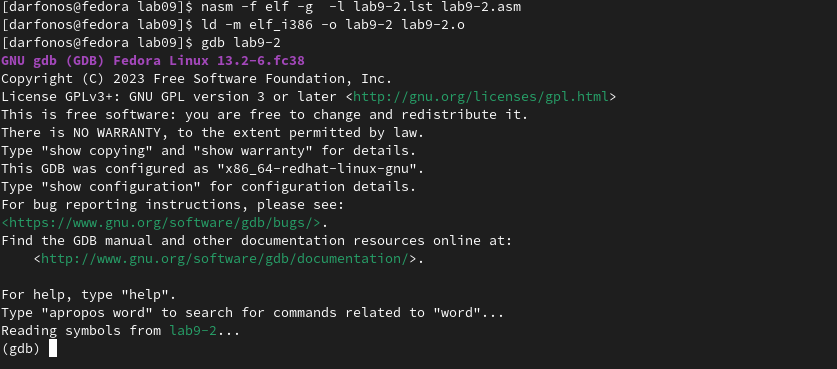


Figure 7: Создание исполняемого файла

**Шаг 8**

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. [[8](#fig:008)]).

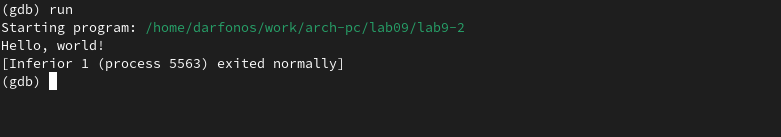


Figure 8: Создание файла

**Шаг 9**

Для более подробного анализа программы, вставляю брэйкпоинт на метку \_start (рис.[[9](#fig:009)]).

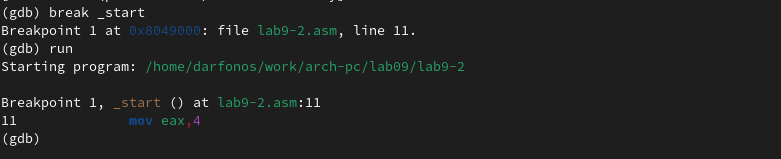


Figure 9: Работа с откладкой gdb

**Шаг 10**

Посмотрим дизассемеблированный код, начиная с этой метки. (рис. [[10](#fig:010)]).

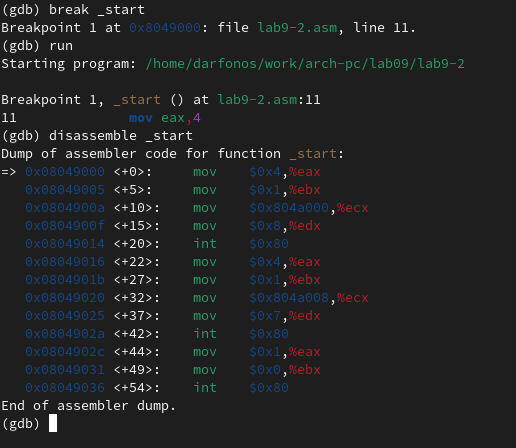


Figure 10: Дисассамблеривоние кода

**Шаг 11**

Так же посмотрим как выглядит дизассемблированный код c синтаксисом Intel (рис. [[11](#fig:011)]).

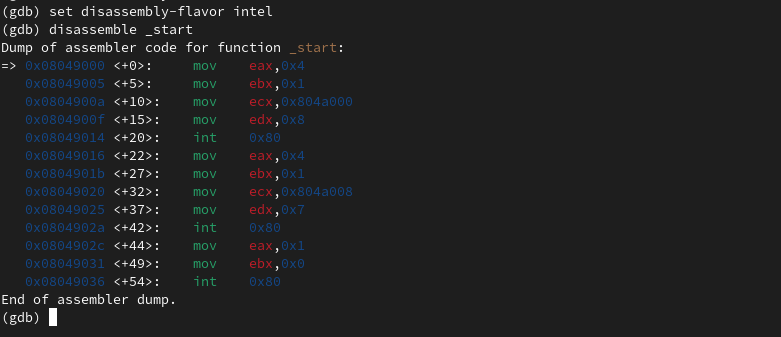


Figure 11: Cинтаксис Intel

* В представлении ATT в виде 16-ричного числа записаны первые аргументы всех комманд, а в представлении intel так записываются адреса вторых аргумантов.

**Шаг 12**

Включим режим псевдографики, с помощью которого отбражается код программы и содержимое регистров (рис. [[12](#fig:012)]).

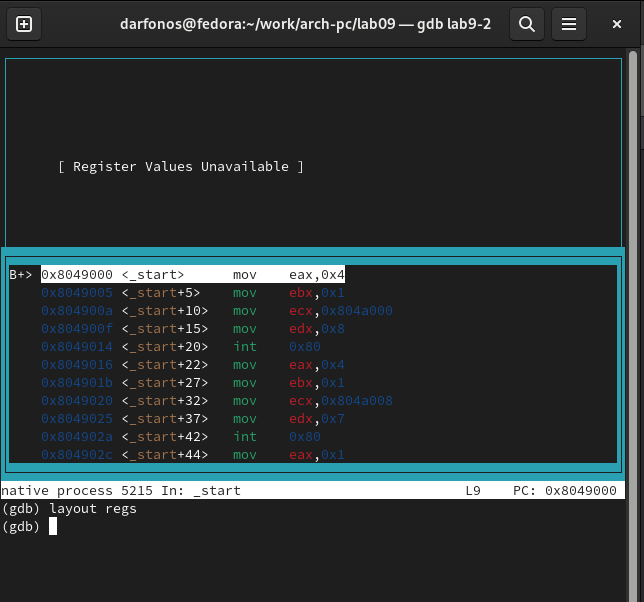


Figure 12: Режим псевдографики

**Шаг 13**

Посмотрим информацию о наших точках останова и сразу добавим еще одну точку .(рис. [[13](#fig:013)]).

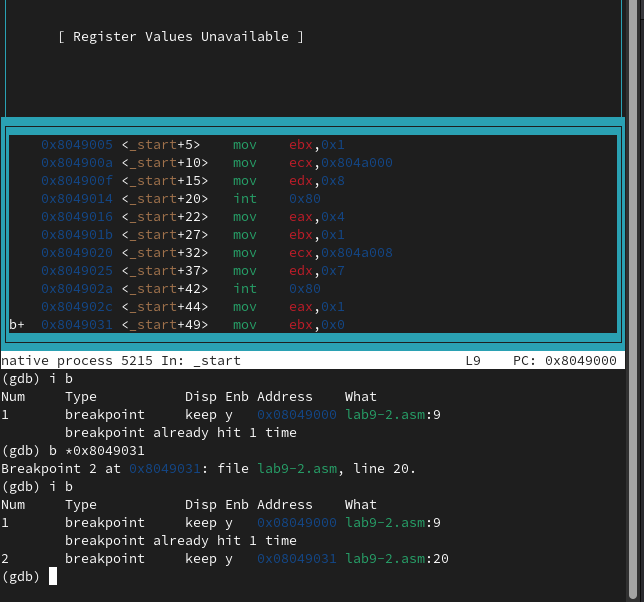


Figure 13: Просмотр точек остонова

**Шаг 14**

Так же можно выводить значения регистров. Делается это командой i r. Псевдографика предствалена на (рис. [[14](#fig:014)]).

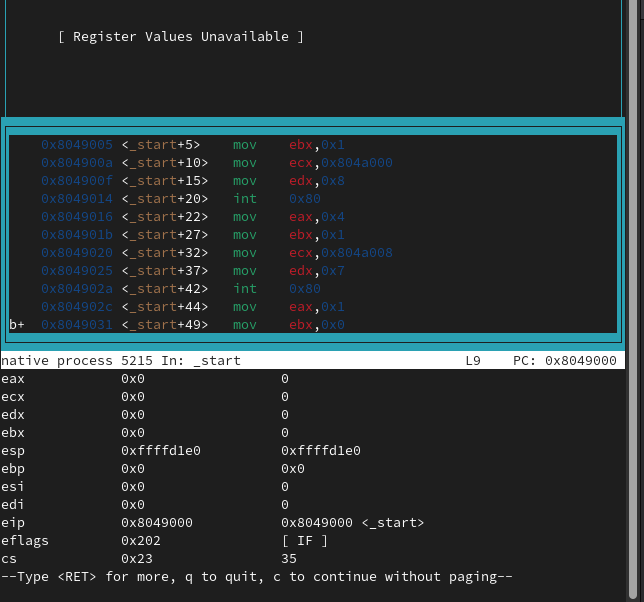


Figure 14: Вывод значений регистров

**Шаг 15**

В отладчике можно вывести текущее значение переменных. Сделать это можно по имени или по адресу: выводим значения переменных msg1 и msg2 (рис. [[15](#fig:015)]).

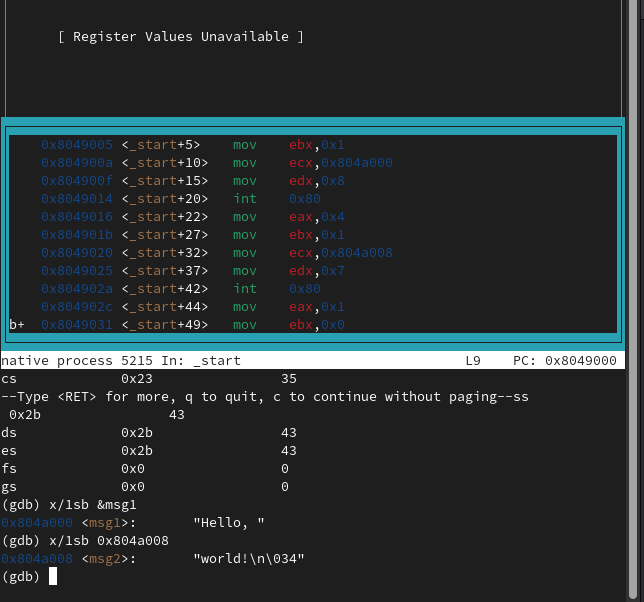


Figure 15: Вывод значений переменных

**Шаг 16**

Так же отладчик позволяет менять значения переменных прямо во время выполнения программы (рис. [[16](#fig:016)]).

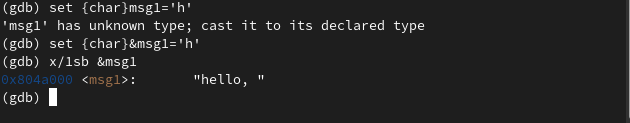


Figure 16: Изменение значений переменных

* Заменяю первый символ ‘H’ на ‘h’

**Шаг 17**

Замененяю первый символ переменной msg2 на символ j. (рис. [[17](#fig:017)]).

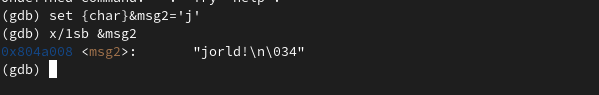


Figure 17: Изменение значений переменных

**Шаг 18**

Выоводить можно так же содержимое регисторов. Выведем значение ebx в разных форматах: строчном, 16-ричном . (рис. [[18](#fig:018)]).

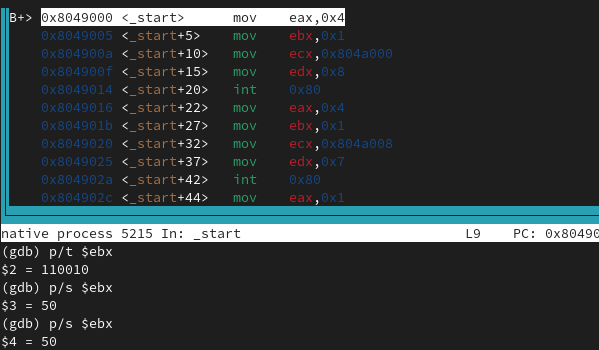


Figure 18: Запуск исполняемого файла

**Шаг 19**

Как и переменным, регистрам можно задавать значения (рис. [[19](#fig:019)]).



Figure 19: Вывод значений переменных

**Шаг 20**

Так же отладчик позволяет менять значения переменных прямо во время выполнения программы (рис. [[20](#fig:020)]).



Figure 20: Вывод значений переменных

* Однако при попытке задать строчное значение, происходит ошибка.

**Завершим работу в gdb командами continue, она закончит выполнение программы, и exit, она завершит сеанс gdb**

**Шаг 21**

Скопируем файл из лабораторной 9, переименуем её и создадим исполняемый файл.(рис. [[21](#fig:021)]).

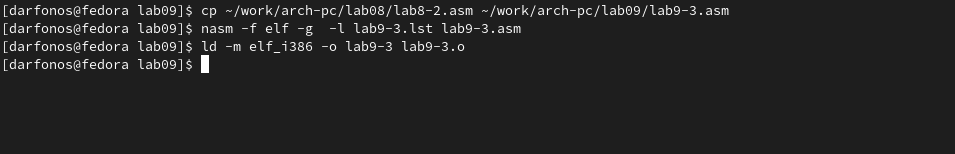


Figure 21: копирование файла

**Шаг 22**

Откроем отладчик и зададим аргументы. (рис. [[22](#fig:022)]).

Figure 22: Запуск исполняемого файла

Figure 22: Запуск исполняемого файла

**Шаг 23**

Создадим точку останова на метке \_start и запустим программу(рис. [[23](#fig:023)]).

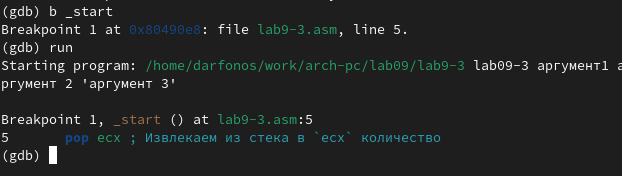


Figure 23: Изменение значений переменных

**Шаг 24**

Посмотрим на содержимое стека, что расположено по адрессу, находящемуся в регистре esp(рис. [[24](#fig:025)]).

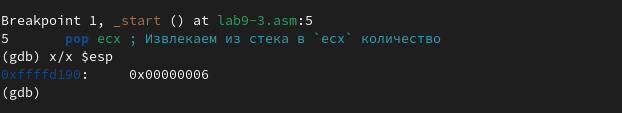


Figure 24: Просмотр содержимого в esp

**Шаг 25**

Далее посмотрим на все остальные аргументы в стеке. Их адреса распологаются в 4 байтах друг от друга(именно столько занимает элемент стека) (рис. [[25](#fig:026)]).

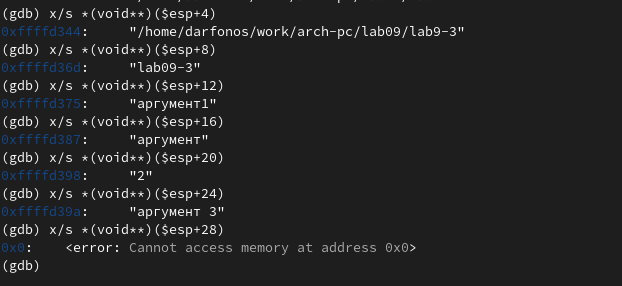


Figure 25: Вывод значений переменных

# 3 Самостоятельная работа

## 3.1 Здание №1

**Шаг 1**

Копирую программу из лабороторной 8 (рис. [[26](#fig:027)]).

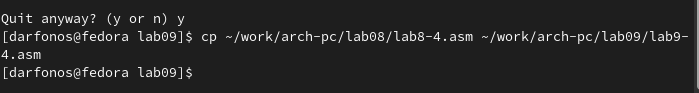


Figure 26: Копирование файла

**Шаг 2**

Изменяю текст программы с использованием подпрограмм (рис. [[27](#fig:028)]).

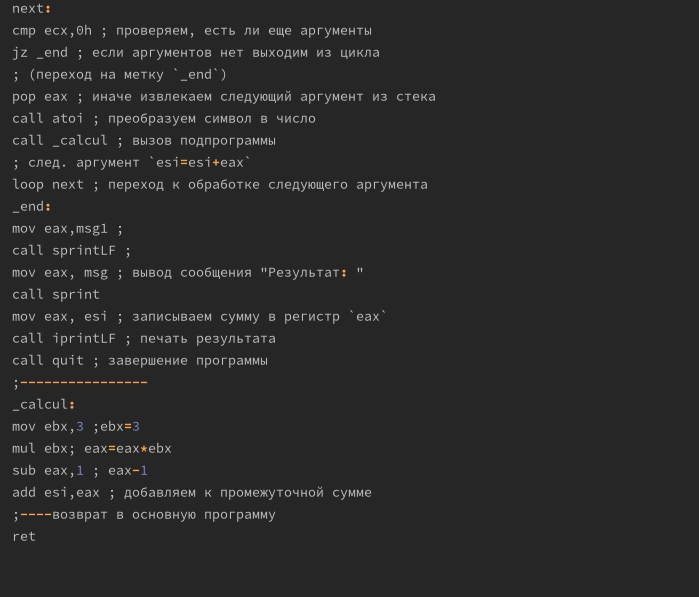


Figure 27: Изменение программы

**Шаг 3**

Создаю исполняемый файл и проверяю работу изменённой программы .(рис. [[28](#fig:029)]).

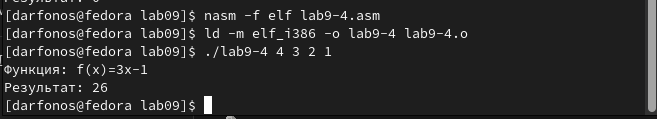


Figure 28: Запуск исполняемого файла

**Программа отработала верно**

## 3.2 Задание №2

**Шаг 1**

Создаю новый файл и вставляю в него программу из листинга (рис. [[29](#fig:030)]).

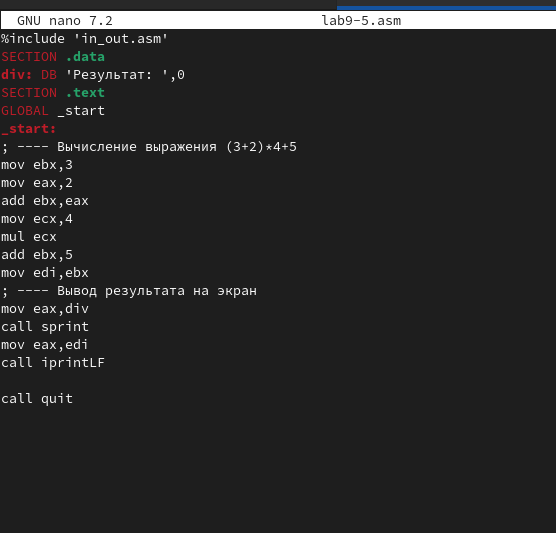


Figure 29: Программа вычисления выражения (3 + 2) \* 4 + 5

**Шаг 2**

Проверяю работу программы и вижу, что в его тексте есть ошибки (рис. [[30](#fig:031)]).

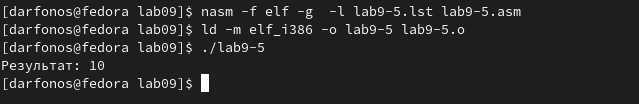


Figure 30: Запуск исполняемого файла

**Шаг 3**

Далее открываю прогрмму в отладчике. Для того, чтобы найти ошибку дисассемблирую программу и добавляю брейкпоинты в основной части программы (рис. [[31](#fig:033)]).



Figure 31: Работа с отладчиком

**Шаг 4**

Запускаю программу до первой точки останова, и проверяю значения регистров.

* Замечаю, что результат сложение записывается в регистр ebx. (рис. [[32](#fig:034)]).

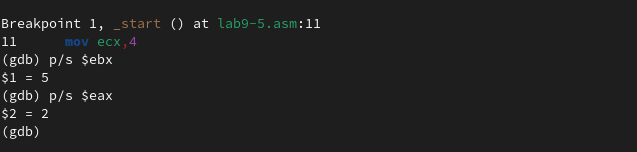


Figure 32: Проверка значений регистров

**Шаг 5**

Перехожу к следующему брейкпоинту и снова проверяю какие значения принимают регистры. (рис. [[33](#fig:035)]).

* Замечаю, что умножение регистра ecx происходит на регистр eax(4\*2), а к регистру ebx плюсуется 5 (5+5) и его значенние записыватся в результат программы.

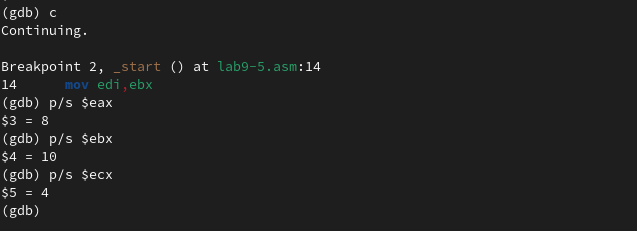


Figure 33: Выявление главных ошибок

**Шаг 6**

Исправляю основные ошибки выявленные с помощью отладчика GDB. (рис. [[34](#fig:036)]).



Figure 34: Исправление ошибок в программе

**Шаг 7**

Создаю исполняемый файл и проверяю работу программы. (рис. [[35](#fig:037)]).

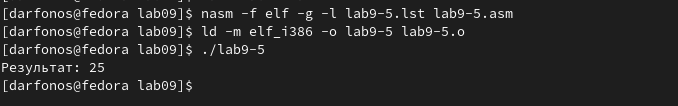


Figure 35: Запуск исполняемого файла

**Программа отработала без ошибок!!**

# 4 Вывод

В результате выполнения лабораторной работы, я научился организовывать код в подпрограммы и познакомился с базовыми функциями отладчика GDB.