Лабораторная работа №9

Дисциплина: Архитектура компьютеров и операционные системы

Чистов Даниил Максимович

Содержание

# 1 Цель работы

Цель работы - Приобретение навыков написания программ с использованием подпрограмм и знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Задание

* Выполнение Лабораторной работы
* Задание для самостоятельной работы

# 3 Выполнение лабораторной работы

## 3.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы, а потом создаю файл lab09-1.asm (рис. [1](#fig:001)).

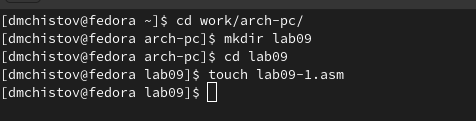


Figure 1: Создание каталога и файлов

Вставляю в созданный файл код (рис. [2](#fig:002)).

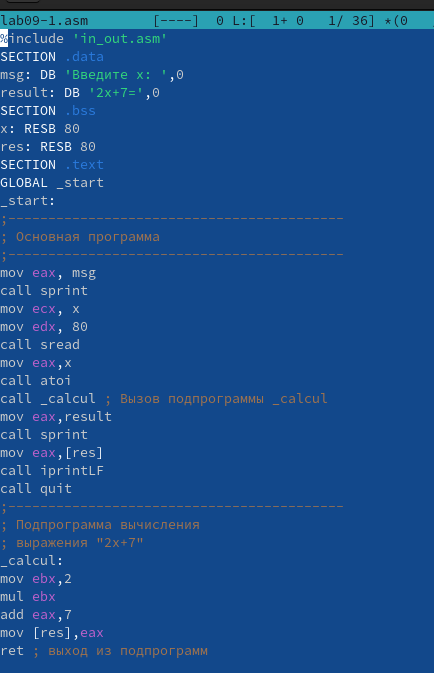


Figure 2: Код программы

Создаю файл и проверяю его работу (рис. [3](#fig:003)).

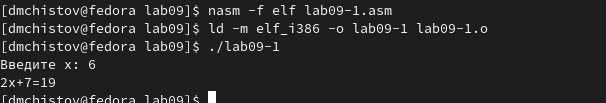


Figure 3: Работа программы

Изменяю код программы, добавив подпрограмму \_subcalcul (рис. [4](#fig:004)).

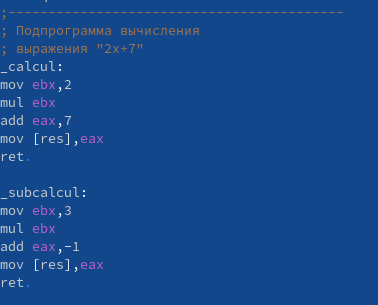


Figure 4: Изменение кода программы

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

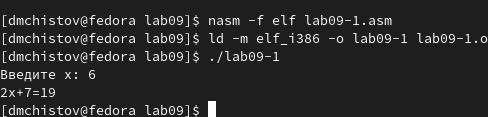


Figure 5: Работа программы

## 3.2 Отладка программам с помощью GDB

Создаю файл lab09-2.asm, а затем вставляю код из листинга 9.2 (рис. [6](#fig:006)).

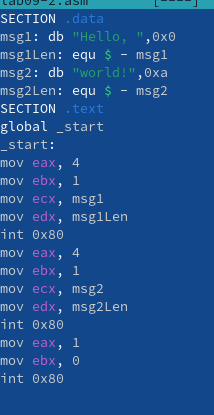


Figure 6: Код программы

Начинаю открывать файл в GDB (рис. [7](#fig:007)).

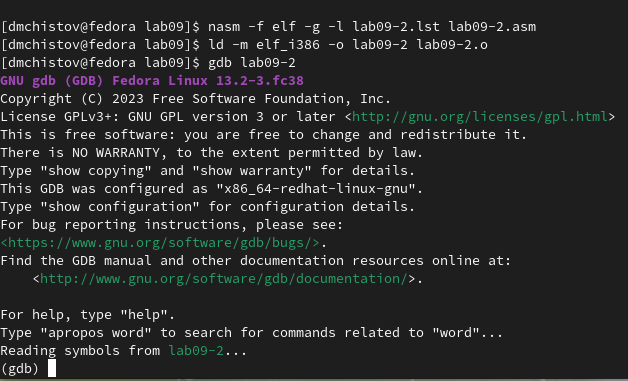


Figure 7: Запуск программы в GDB

Проверяю работу программы (рис. [8](#fig:008)).

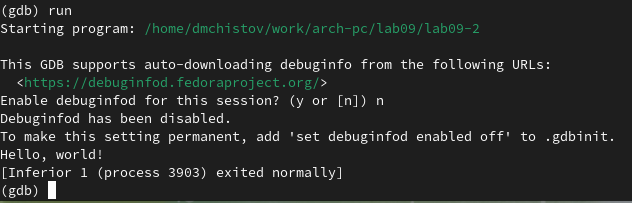


Figure 8: Программа работает успешно

Ставлю метку у кода \_start (рис. [9](#fig:009)).

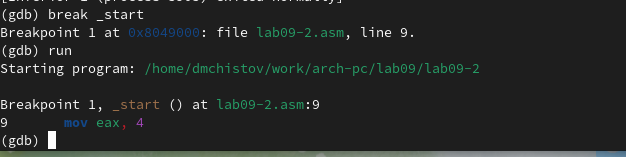


Figure 9: Метка у \_start

Дисассимилирую код с помощью команды diassemble (рис. [10](#fig:010)).



Figure 10: Диассимилированный код

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. [11](#fig:011)).

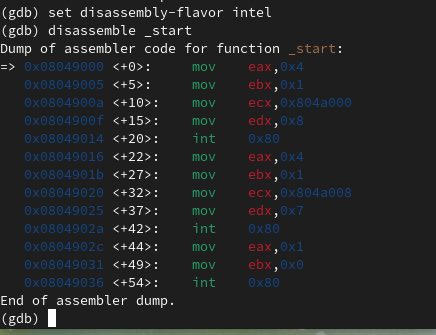


Figure 11: Отображение команд

Имена регистров в режиме ATT начинаются с символа %, а имена операндов – с $, в то время как в синтаксисе Intel используется более привычный формат.

Включаю специальные режимы отображения (рис. [12](#fig:012)), (рис. [13](#fig:013)).

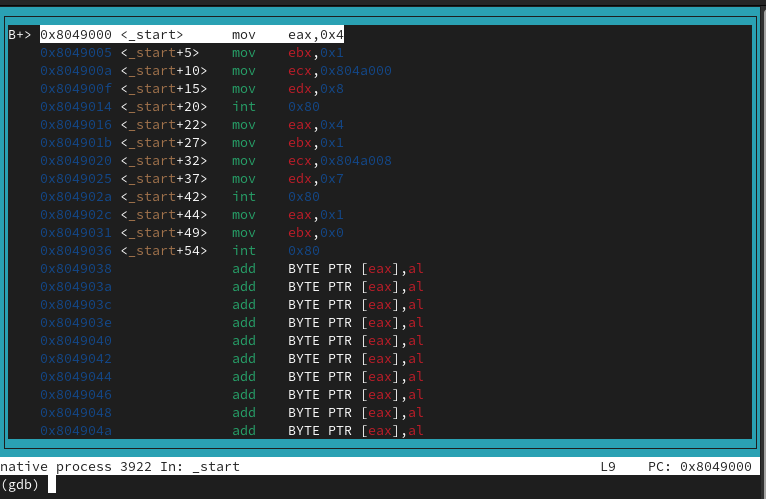


Figure 12: Особое отображение asm

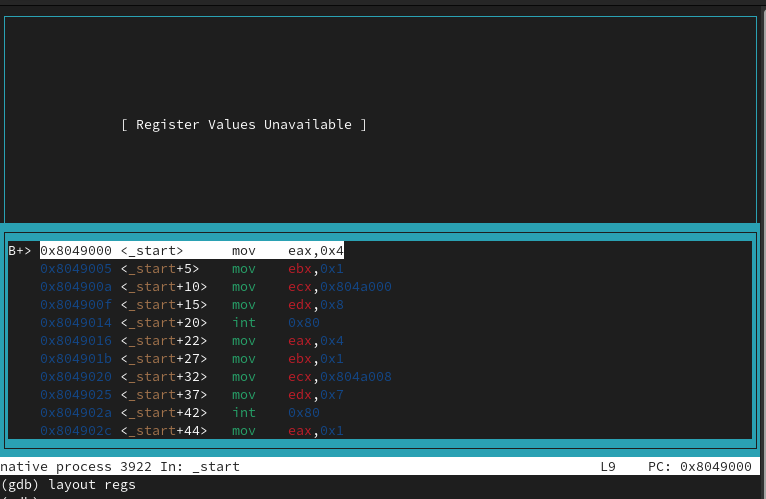


Figure 13: Особое отображение asm

## 3.3 Добавление точек останова

Проверяю наличие точек останова командой info breakpoints (рис. [14](#fig:014)).

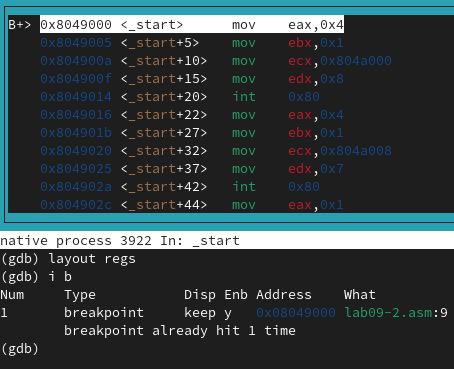


Figure 14: Отображение точек останова

Определяю адрес инструкции mov ebx,0x0, а затем устанавливаю точку останова с помощью команды break (рис. [15](#fig:015)).

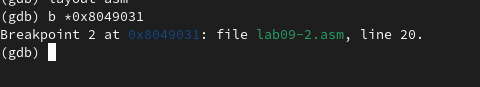


Figure 15: Установка точки останова

## 3.4 Работа с данными программы в GDB

С помощью команды si выполняю пять инструкций, затем проверяю значения каких регистров меняются (рис. [16](#fig:016)).

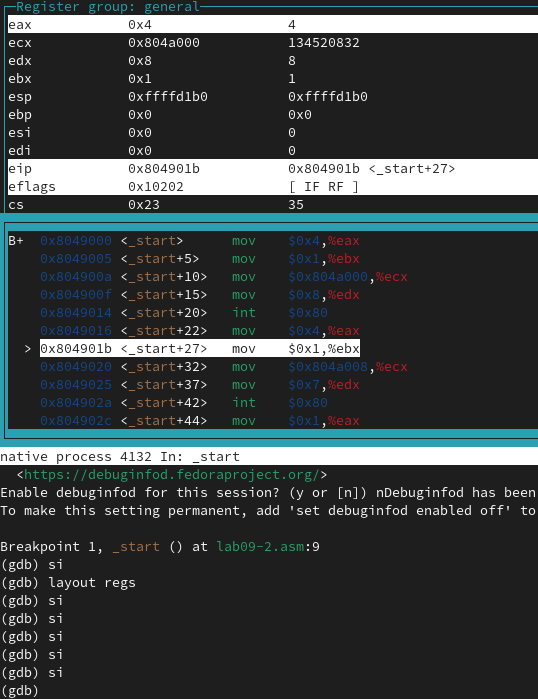


Figure 16: Отображение точек останова

Значения регистров eax, ecx, edx и ebx были изменены.

Просматриваю значения регистров с помощью команды info registers (рис. [17](#fig:017)).

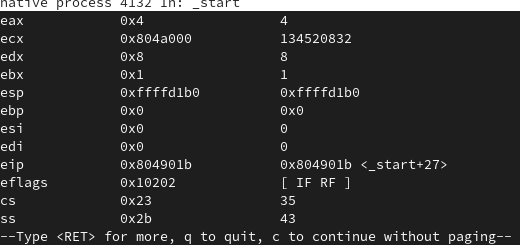


Figure 17: Значения регистров

Просматриваю значение переменной msg1 и msg2 (рис. [18](#fig:018)).

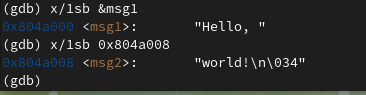


Figure 18: Значения переменных msg1 и msg2

Изменения значений переменных msg1 и msg2 с помощью команды set (рис. [19](#fig:019)).

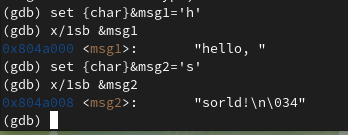


Figure 19: Изменения значений msg1 и msg2

Теперь вывожу значения регистра edx в различных форматах (рис. [20](#fig:020)).

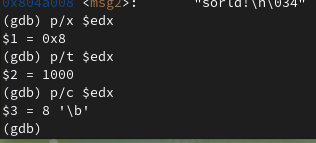


Figure 20: Различные форматы значений регистра edx

Команда вывода p/s $ebx различается в том, что при первой мы преобразуем символ в его текстовое представление, в то время как при второй - числовое значение в строковом виде остается неизменным.

Завершаю работу программы с помощью команды c, после чего выхожу из GDB с помощью команды q (рис. [21](#fig:021)).

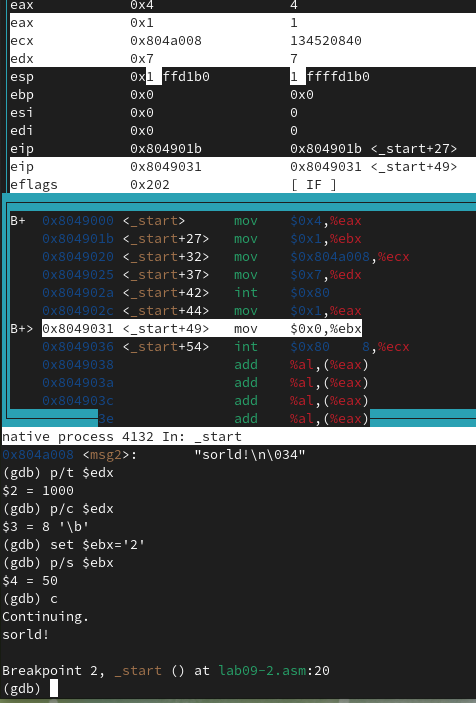


Figure 21: Завершение работы программы

## 3.5 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm, а затем компилирую его (рис. [22](#fig:022)).

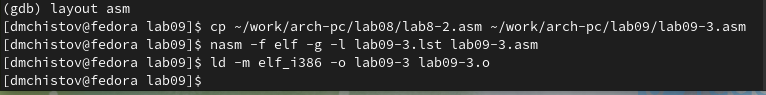


Figure 22: Копирование файла lab8-2

Загружаюсь в gdb перед этим указав аргументы (рис. [23](#fig:023)).

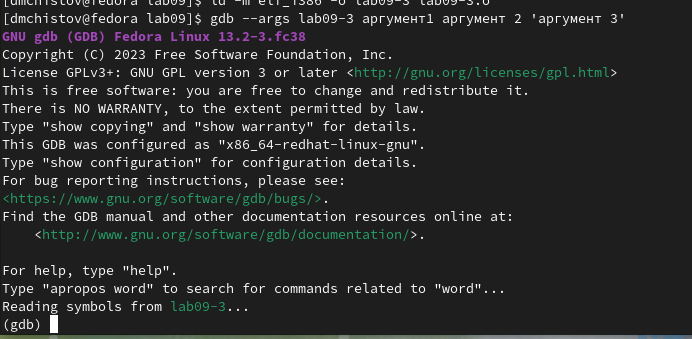


Figure 23: Загрузка в gdb

Просматриваю позиции стека по адресу [esp+4], [esp+8], [esp+12] и т.д. (рис. [24](#fig:024)).

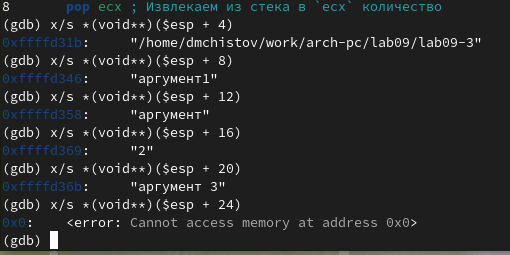


Figure 24: Просмотр позиций стека

Шаг изменения адреса составляет 4, поскольку количество аргументов в командной строке также равно 4.

# 4 Задание для самостоятельной работы

## 4.1 Задание 1

Копирую программу из задания 1 лабораторной работы №8, а затем изменяю её, реализовав вычисление значения функции f(x) как подпрограмму (рис. [25](#fig:025)).

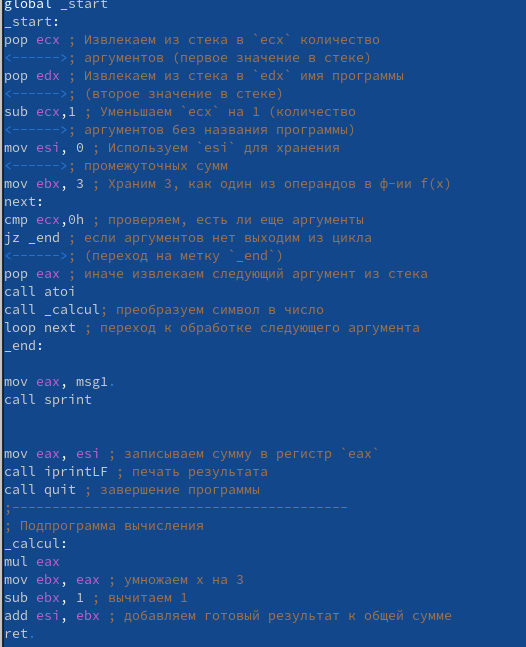


Figure 25: Код программы

Проверяю работу программы (рис. [26](#fig:026)).

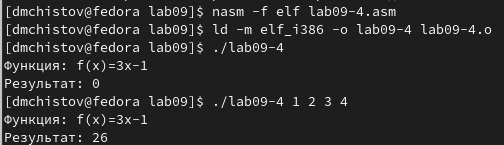


Figure 26: Программа работает успешно!

Код программы: %include ‘in\_out.asm’ SECTION .data msg1 db “Функция: f(x)=3x-1”, 0x0A msg2 db “Результат:” SECTION .text global \_start \_start: pop ecx ; Извлекаем из стека в ecx количество ; аргументов (первое значение в стеке) pop edx ; Извлекаем из стека в edx имя программы ; (второе значение в стеке) sub ecx,1 ; Уменьшаем ecx на 1 (количество ; аргументов без названия программы) mov esi, 0 ; Используем esi для хранения ; промежуточных сумм mov ebx, 3 ; Храним 3, как один из операндов в ф-ии f(x) next: cmp ecx,0h ; проверяем, есть ли еще аргументы jz \_end ; если аргументов нет выходим из цикла ; (переход на метку \_end) pop eax ; иначе извлекаем следующий аргумент из стека call atoi call \_calcul; преобразуем символ в число loop next ; переход к обработке следующего аргумента \_end:

mov eax, msg1 call sprint

mov eax, esi ; записываем сумму в регистр eax call iprintLF ; печать результата call quit ; завершение программы ;—————————————— ; Подпрограмма вычисления \_calcul: mul eax mov ebx, eax ; умножаем x на 3 sub ebx, 1 ; вычитаем 1 add esi, ebx ; добавляем готовый результат к общей сумме ret

## 4.2 Задание 2

Открываю код программы из листинга 9.3 (рис. [27](#fig:027)).

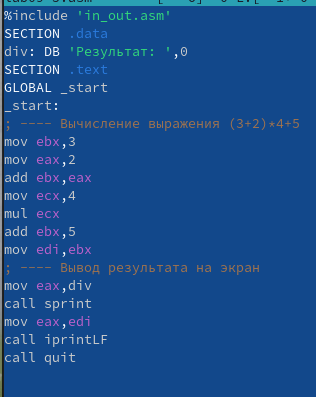


Figure 27: Код программы

Проверяю работу программы, результат должен быть равен 25 (рис. [28](#fig:028)).

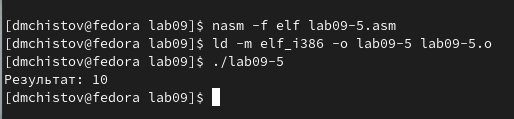


Figure 28: Программа работает некорректно!

Программа выводит неправильный результат, нужно искать ошибку. Компилирую код для работы с gdb, затем запускаю её (рис. [29](#fig:029)).

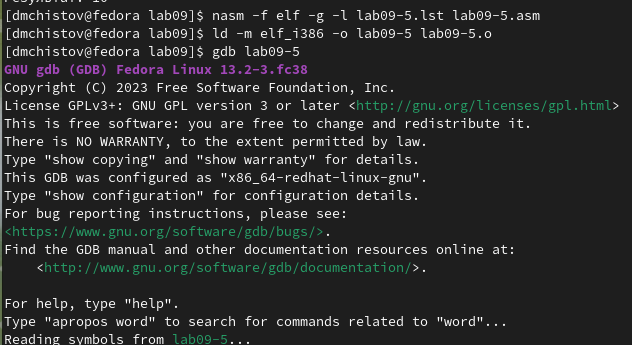


Figure 29: Подготовка программы

Ставлю точку останова с момента \_start (рис. [30](#fig:030)).

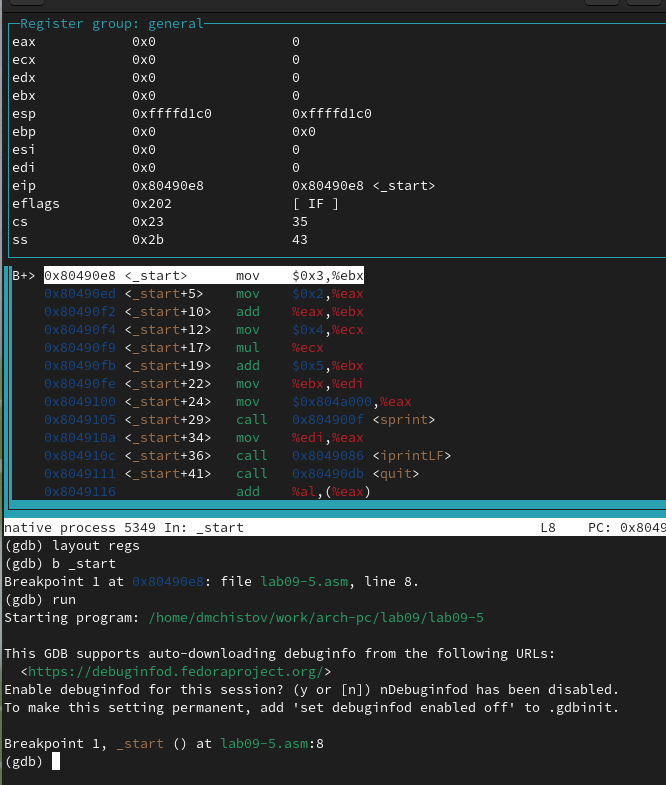


Figure 30: Подготовка программы

Начинаю следить за значениями переменных. На (рис. [31](#fig:031)), (рис. [32](#fig:032)) и (рис. [33](#fig:033)) всё идёт хорошо.

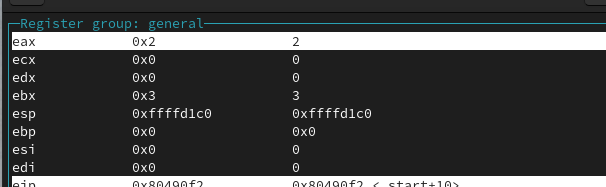


Figure 31: Изменение переменной eax

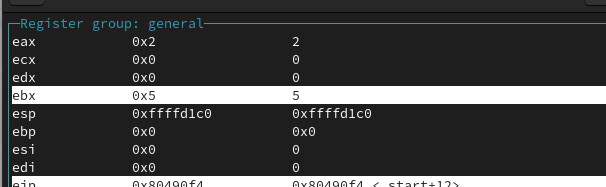


Figure 32: Изменение переменной ebx

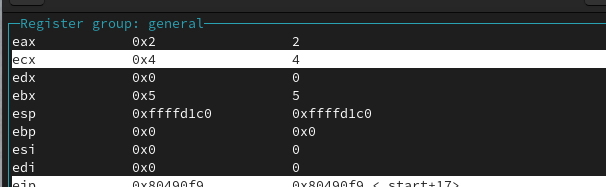


Figure 33: Изменение переменной ecx

А вот в этом моменте явно происходит ошибка, происходит перемножение значений переменных eax и ecx, а должно ecx и ebx (рис. [34](#fig:034)).

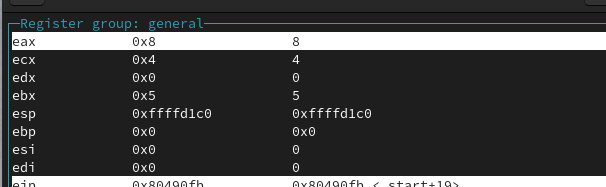


Figure 34: Ошибка в коде

Чтобы исправить ошибку, мы добавляем после add ebx, eax строку mov eax, ebx и заменяем ebx на eax в инструкциях add eax, 5 и mov edi, eax (рис. [35](#fig:035)).

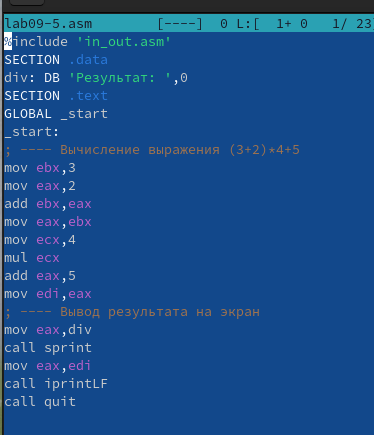


Figure 35: Исправленный код

Проверяю работу программы (рис. [36](#fig:036)).

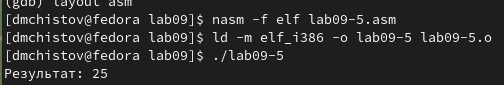


Figure 36: Программа работает успешно!

Код программы: %include ‘in\_out.asm’ SECTION .data div: DB ‘Результат:’,0 SECTION .text GLOBAL \_start \_start: ; —- Вычисление выражения (3+2)\*4+5 mov ebx,3 mov eax,2 add ebx,eax mov eax,ebx mov ecx,4 mul ecx add eax,5 mov edi,eax ; —- Вывод результата на экран mov eax,div call sprint mov eax,edi call iprintLF call quit

# 5 Выводы

Благодаря выполнению работы, я научился приобретению навыков написания программ с использованием подпрограмм и познакомился с методами отладки при помощи GDB и его основными возможностями.

# Список литературы

[Лабораторная работы №9](https://esystem.rudn.ru/pluginfile.php/2089096/mod_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%969.%20%D0%9F%D0%BE%D0%BD%D1%8F%D1%82%D0%B8%D0%B5%20%D0%BF%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B.%20%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA%20..pdf)