Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB

Колонтырский Илья Русланович

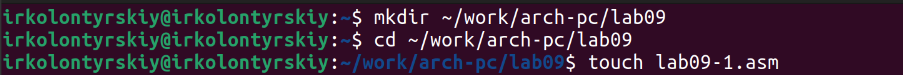
Содержание

# 1 Цель работы

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

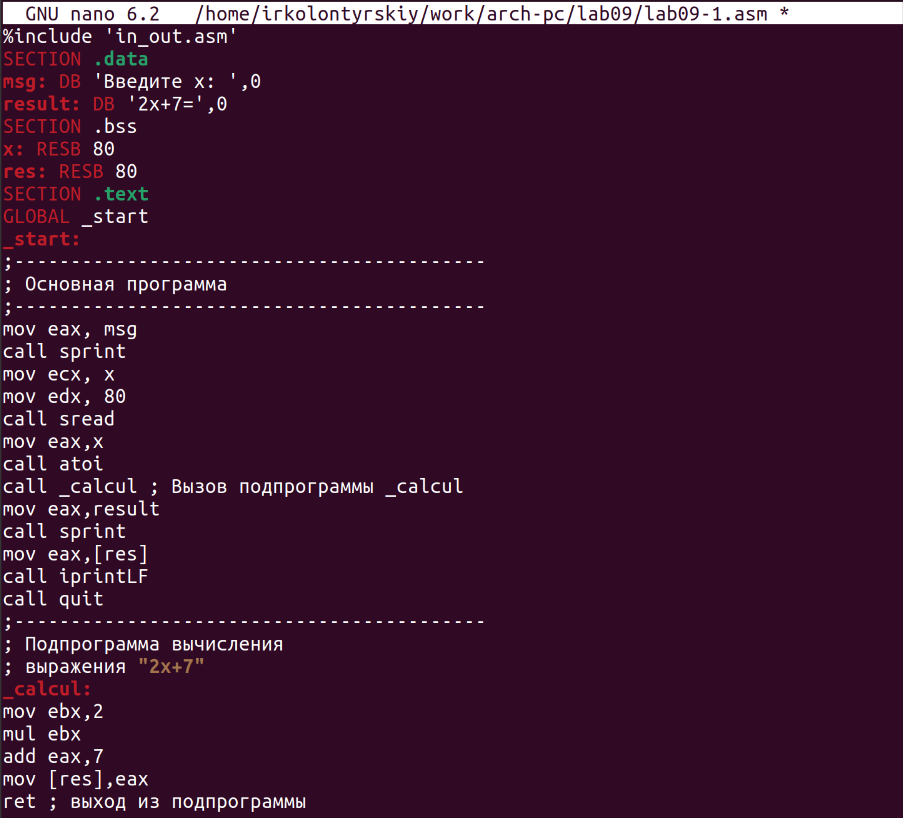
# 2 Выполнение лабораторной работы

Создадим папку и файл lab9-1.asm (Рис. 2.1)



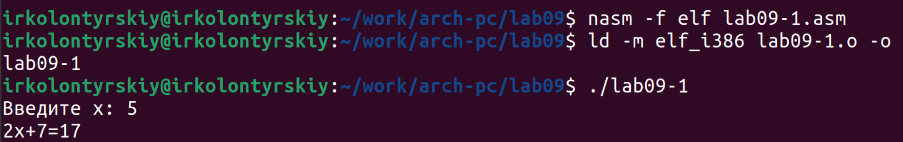
Создание папки и файла lab9-1.asm

Вставим в файл lab9-1.asm предложенный код (Рис. 2.2)



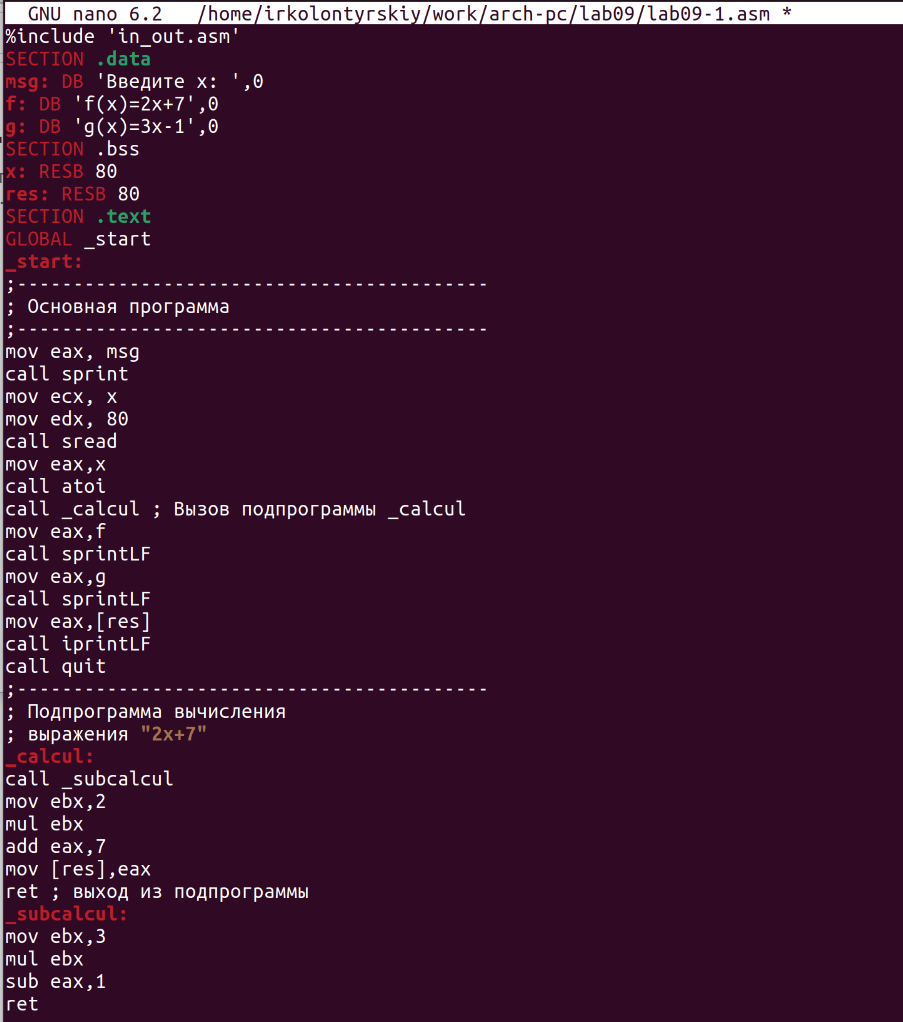
Вставка кода

Скомпилируем программу и посмотрим на результат (Рис. 2.3)



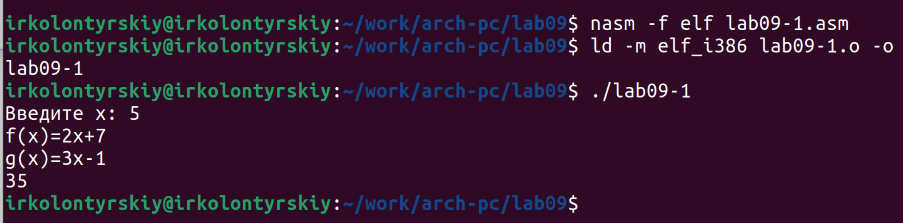
Компиляция программы и запуск

Изменим программу, добавив подпрограмму, которая вычисляет g(x), после чего выводит результат вычисления f(g(x)) (Рис. 2.4)



Изменение файла

Скомпилируем программу и посмотрим на результат (Рис. 2.5)



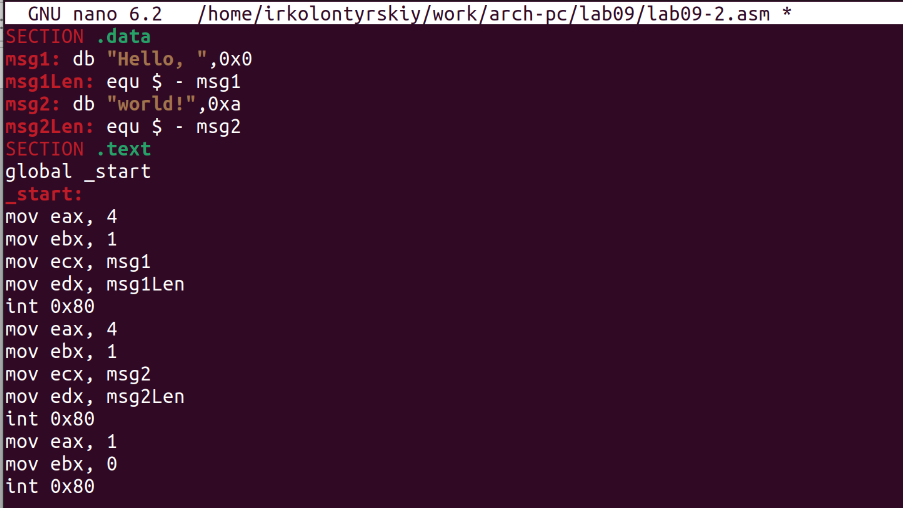
Компиляция программы и повторный запуск

Создадим ещё один файл (Рис. 2.6)

Создание файла lab9-2.asm

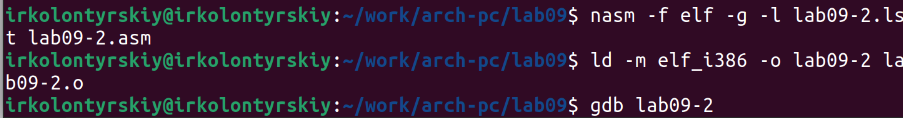
Создание файла lab9-2.asm

Вставим файл следующий код (Рис. 2.7)



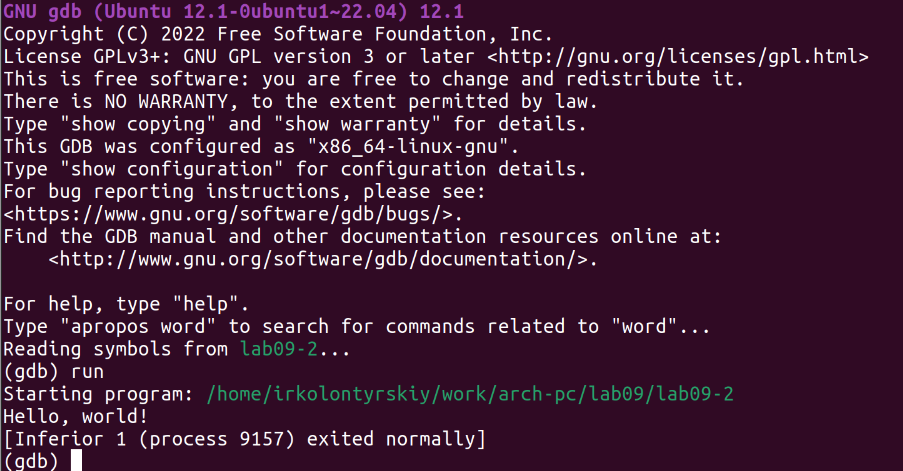
Вставка кода

Скомпилируем программу для отладки (с аргументом -g) и загрузим в gdb (Рис. 2.8)



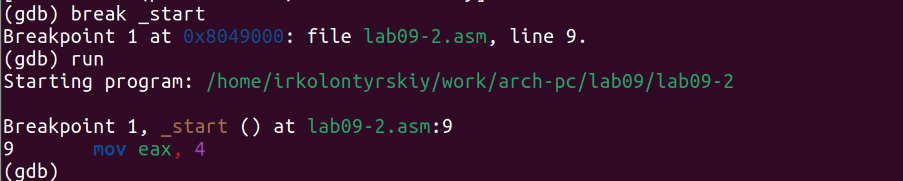
Компиляция файла для отладки

В gdb запустим её с помощью run (Рис. 2.9)



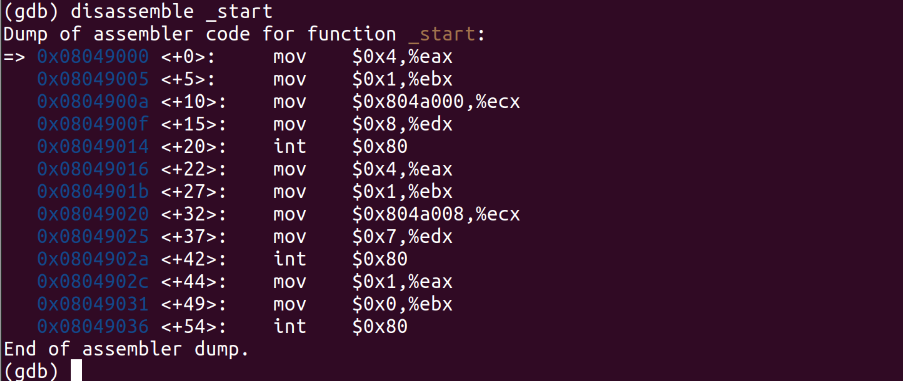
Запуск программы

На метке \_start создадим брейкпоинт (Рис. 2.10)



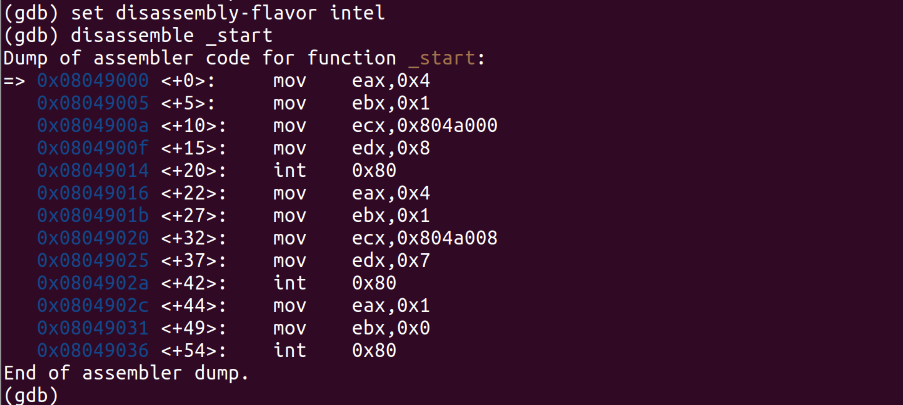
Создание брейкпоинта

Теперь дизассемблируем программу (Рис. 2.11)



Дизассемблирование

Теперь сделаем вывод на синтаксисе intel (Рис. 2.12)



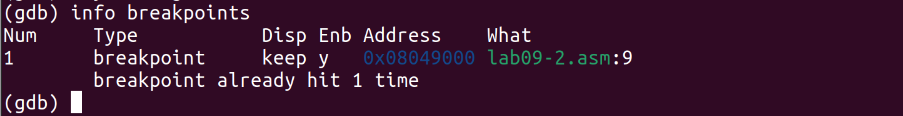
Включение синтаксиса intel

Включим отображения кода и регистров (Рис. 2.13)

Включение отображения кода и регистров

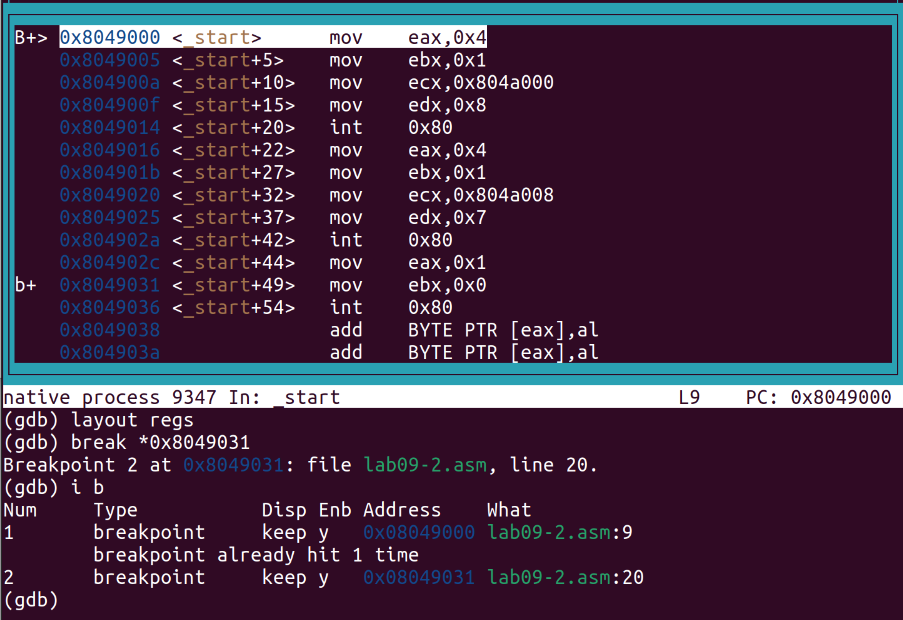
Включение отображения кода и регистров

Выведем инормацию о брейкпоинтах (Рис. 2.14)



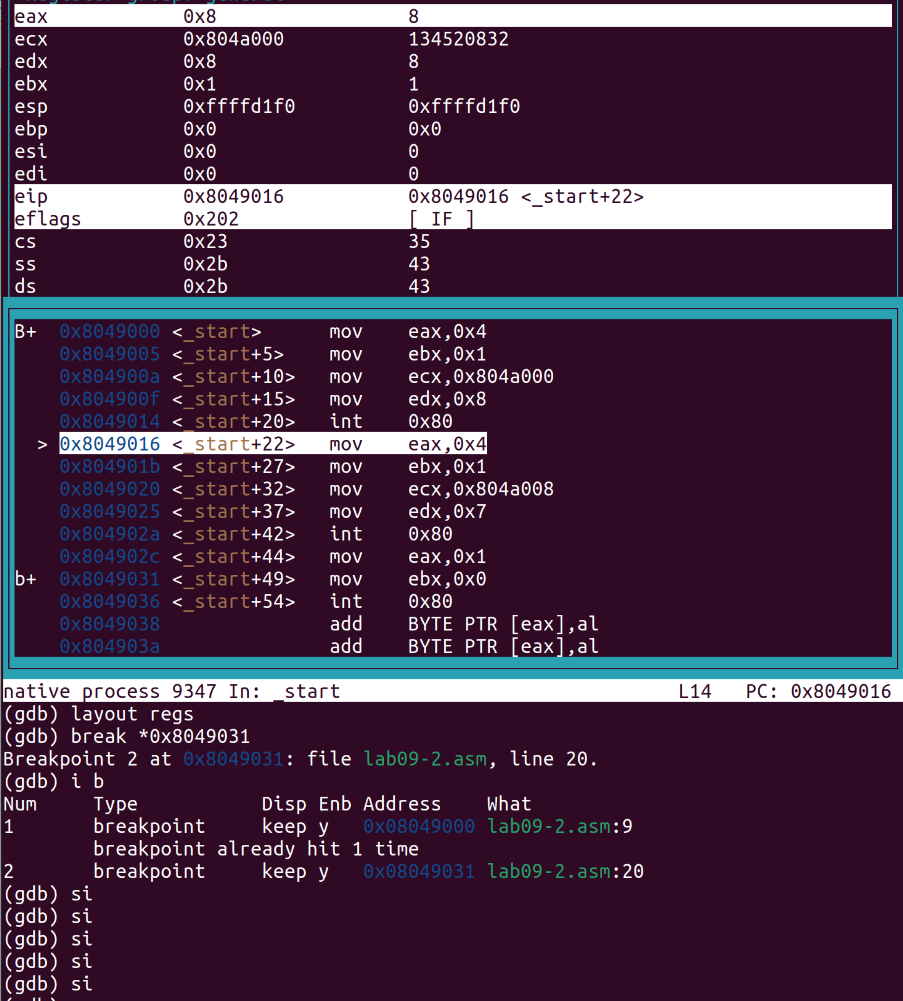
Вывод информации о брейкпоинтах

Теперь создадим брейкпоинт не по имени, а по адресу, а также выведем информацию обо всех брейкпоинтах (Рис. 2.15)



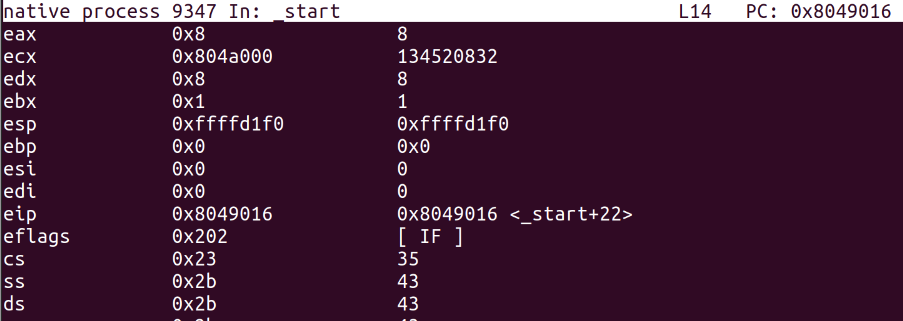
Создание брейкпоинта по адресу и вывод информации о всех брейкпоинтах

Теперь выполним команду si 5 раз, выполнив тем самым последовательно 5 строчек кода (Рис. 2.16)



Использование si

За 5 строк поменялись значения eax, ecx, edx и ebx регистров. Выведем информацию о значениях регистров (Рис. 2.17)



Вывод значений регистров

Выведем значение переменной (Рис. 2.18)

Вывод переменной

Вывод переменной

Выведем теперь значение переменной по адресу (Рис. 2.19)

Вывод переменной по адресу

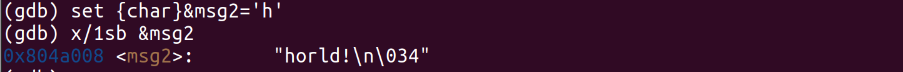
Вывод переменной по адресу

Изменим значение переменной и выведем его (Рис. 2.20)



Изменение переменной и вывод значения

Изменим вторую переменную и выведем её (Рис. 2.21)



Изменение второй переменной и вывод значения

Выведем значение регистра edx в строковом, двоичном, шестнадцатиричном видах (Рис. 2.22)



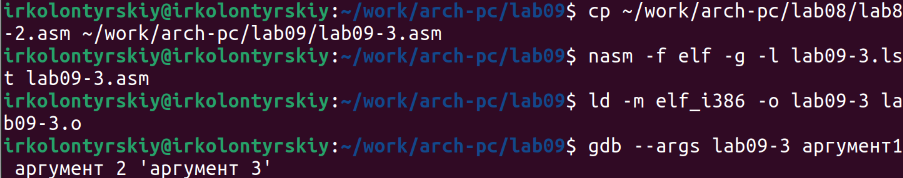
Вывод значения регистра

Изменим теперь значение регистра (Рис. 2.23)



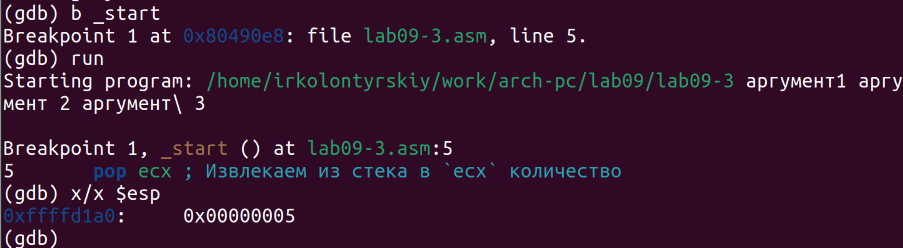
Изменение регистра

Мы записали в регистр 2 разных значения. В первом случае это строка, а во втором - число. Выйдем из отладчика и скопируем файл из прошлой работы, соберм его и вгрузим в отладчик (Рис. 2.24)



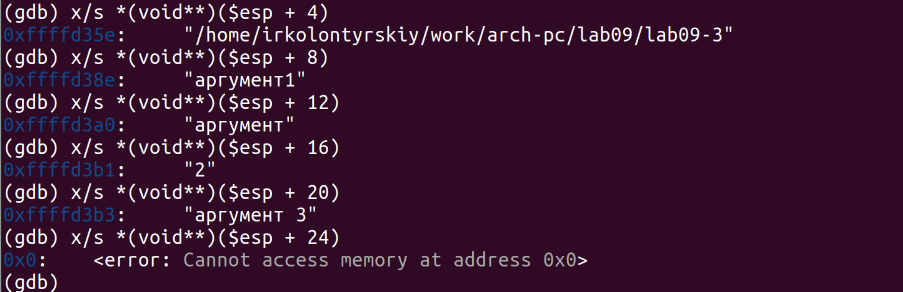
Копирование файла из прошлой работы и выгрузка в gdb

Создадим брейкпоинт и запустим его, после чего выведем значение регистра esp (Рис. 2.25)



Создание брейкпоинта и запуск, вывод значения esp

Выведем значение элементов стека (Рис. 2.26)

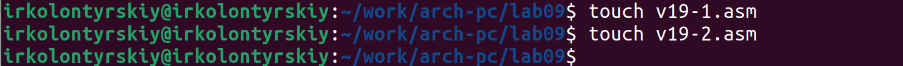


Вывод всех значений в стеке

Поскольку под каждый элемент стека выделяется 4 байта, мы должны выводить элементы с шагом в 4

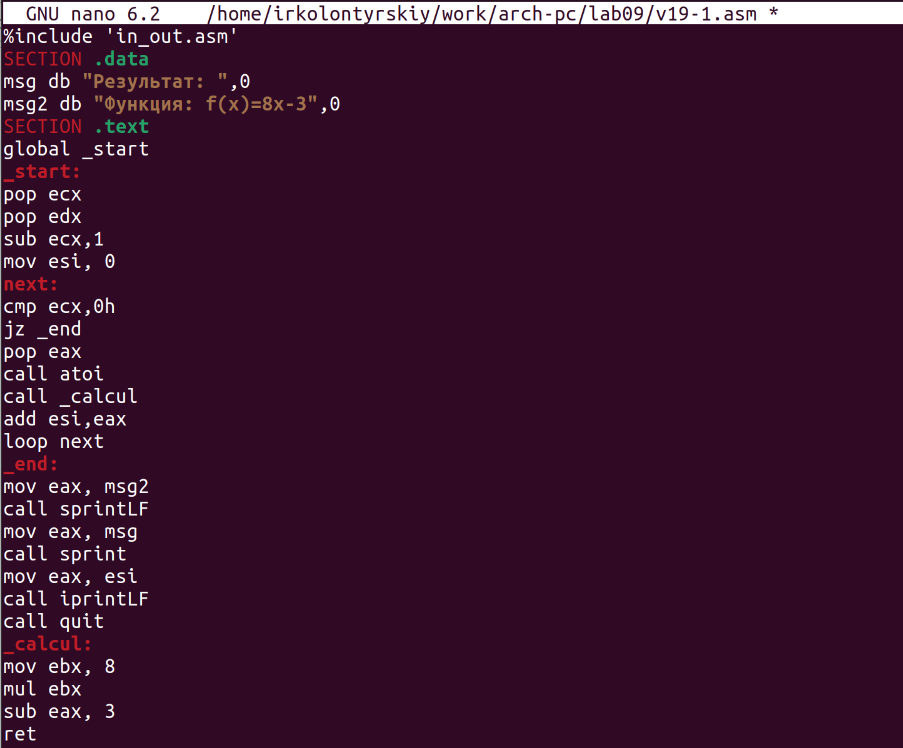
Самостоятельная работа

Создадим файлы для самостоятельной работы (Рис. 2.27)



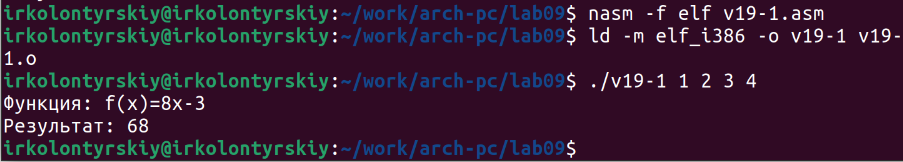
Создание файлов

В первом файле нам нужно переписать файл первой самостоятельной работы прошлой лабораторной работы с использованием подпрограмм (Рис. 2.28)



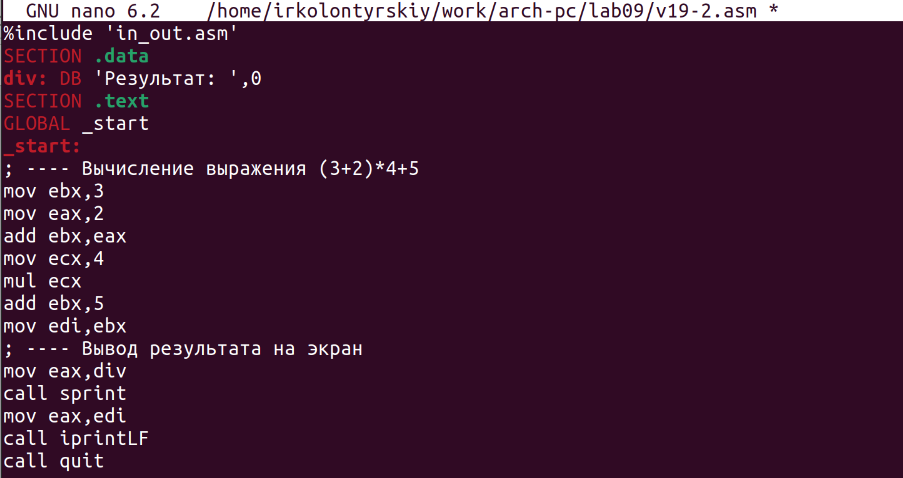
Редактирование программы

Скомпилируем файл и проверим его работу (Рис. 2.29)



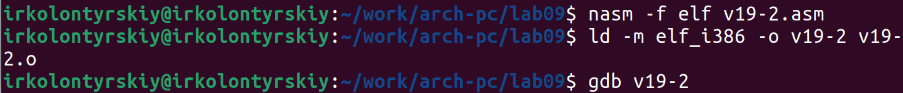
Сборка и проверка работы программы

Вставим во второй код предложенный код (Рис. 2.30)



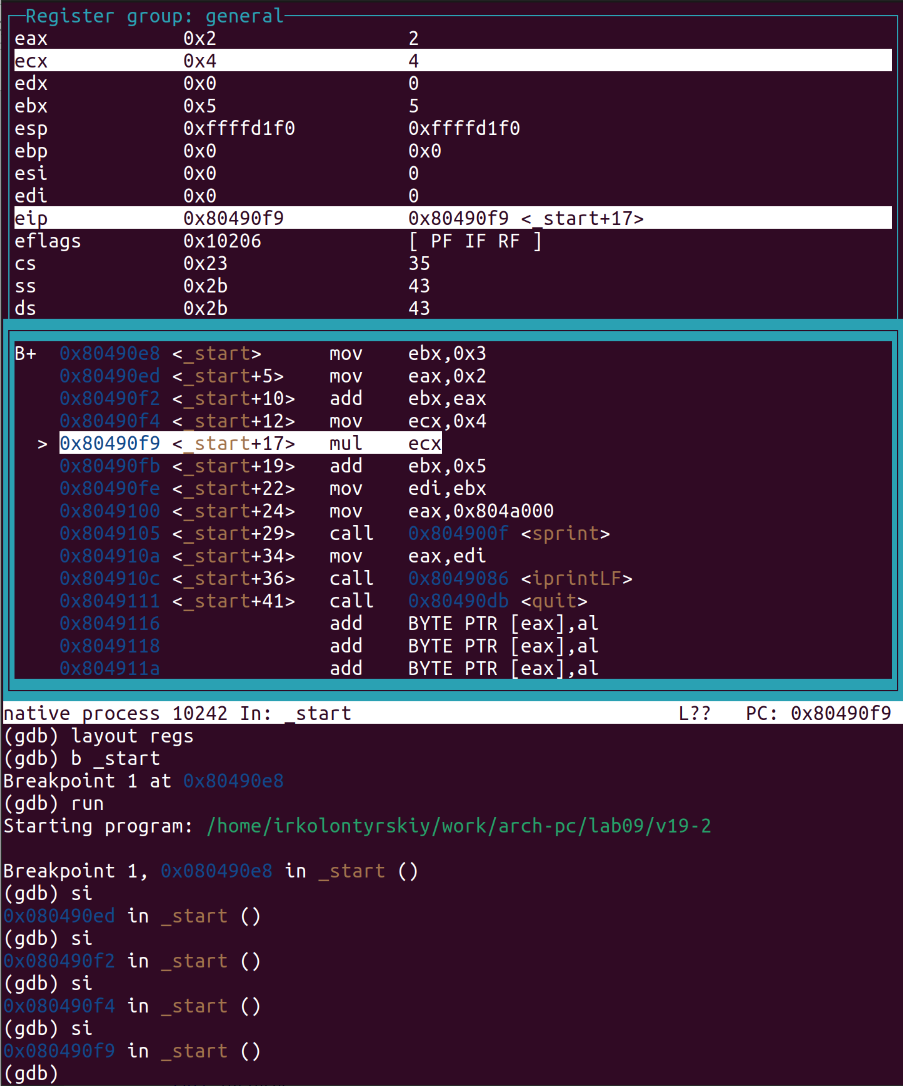
Вставка кода

Соберём его и вставим в gdb (Рис. 2.31)



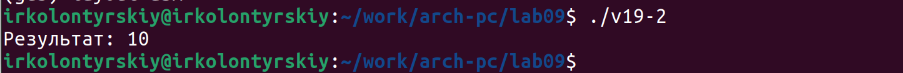
Сборка программы и вгрузка в gdb

Начнём построчно выполнять код (Рис. 2.32):



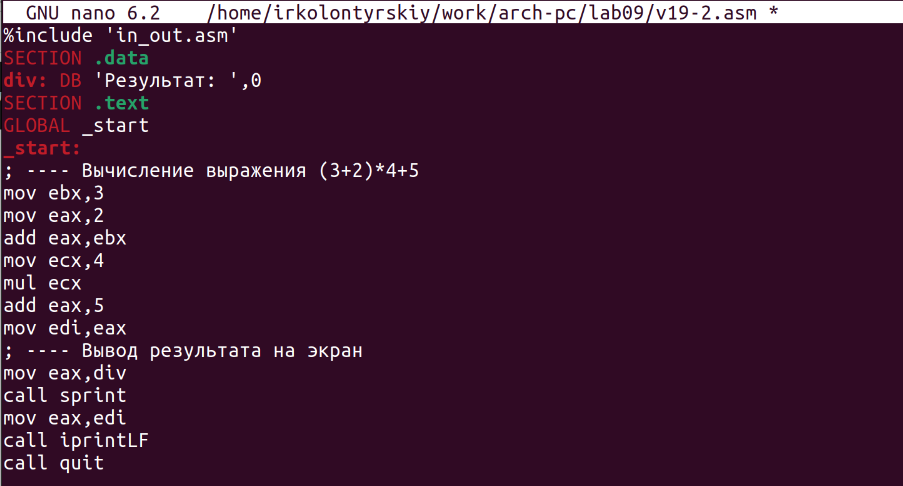
Построчное выполнение кода

Видно, что при умножении результат сохраняется в eax, хотя кодом подразумавается сохранение в ebx. Запустим код и убедимся, что он работает неверно (Рис. 3.33)



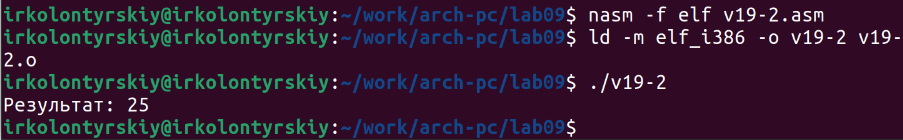
Проверка программы

Изменим его так, чтобы умножалось нужное нам значение, а не то, что хранится в регистре eax изначально (Рис. 2.24)



Изменение кода

Проверим правильность его работы, скомпилировав его и запустив (Рис. 2.35)



Компиляция кода и проверка работы

Теперь код работает правильно

# 3 Выводы

Были получены навыки работы с adb, а также получены навыки работы с подпрограммами