Отчёта по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Павлюченков Сергей Витальевич

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

* Реализация циклов в NASM Создаю каталог для программ лабораторной работы № 9, перехожу в него и создаю файл lab9-1.asm (рис. [1](#fig:001)).

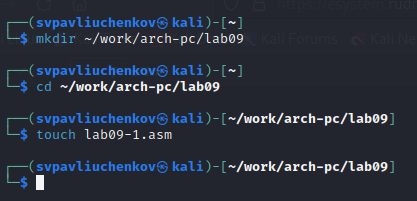


Figure 1: Создание первого файла

Ввожу в файл lab9-1.asm текст программы из листинга 9.1 (рис. [2](#fig:002)).

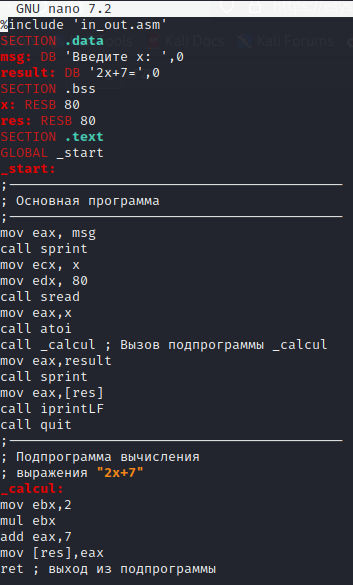


Figure 2: Программа из листинга 9.1

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

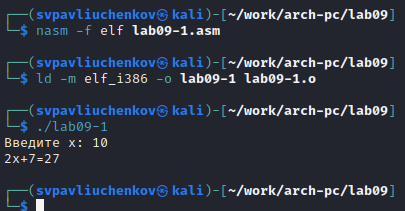


Figure 3: Запуск исполнительного файла

Эта программа считает и выводит выражение , в котором x вводится пользователем.

Изменяю текст программы добавляя подпрограмму subcalcul(рис. [4](#fig:004)).

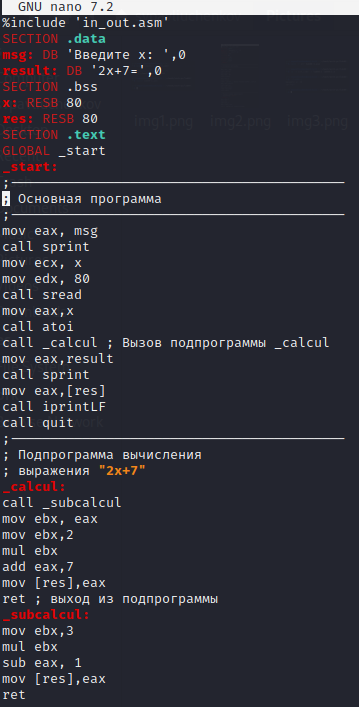


Figure 4: Измененная программа

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

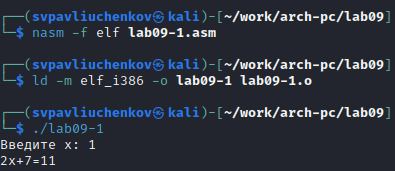


Figure 5: Запуск исполнительного файла

Программа работает корректно(g(x)=, f(g(x)) = )

* Отладка программам с помощью GDB

Cоздаю файл lab9-2.asm (рис. [6](#fig:006))

Figure 6: Создание файла

Figure 6: Создание файла

Ввожу в файл lab9-2.asm текст программы из листинга 9.2 (рис. [7](#fig:007)).

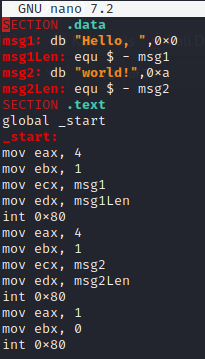


Figure 7: Программа из листинга 9.2

Создаю исполнительный файл и загружаю его в отладчик GDB (рис. [8](#fig:008)).

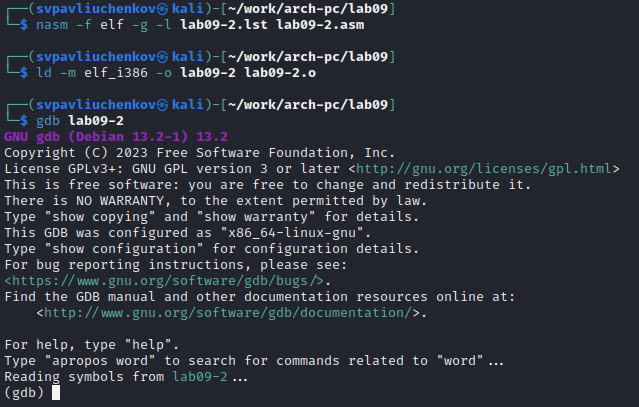


Figure 8: Создание и загрузка исполнительного файла

Проверяю работу программы, запуская её в оболочке GDB (рис. [9](#fig:009)).

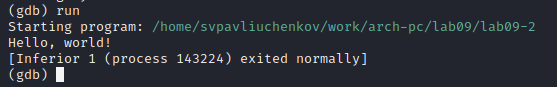


Figure 9: Запуск программы в отладчике

Устанавливаю брейкпоинт и опять запускаю программу (рис. [10](#fig:010)).

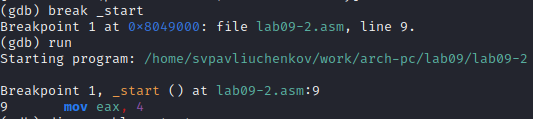


Figure 10: Запуск программы в отладчике с брейкпоинтом

Смотрю дисассимилированный код программы с помощью команды disassemble начиная с метки start (рис. [11](#fig:011)).



Figure 11: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel

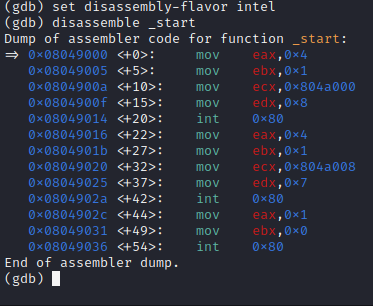


Figure 12: Дисассимилированный код программы с Intel’овским синтаксисом

В режиме ATT для обозначения регистров используется префикс %, а в режиме Intel префиксы для регистров не используются, они просто обозначаются именами регистров. В режиме ATT для обозначения констант используется префикс $, а в режиме Intel префикс $ для обозначения констант не требуется, константы просто записываются как числа или адреса. При выводе дисассимилированного кода в режиме ATT сначала идет константа, и потом регистр, а при выводе в режиме Intel сначала регистр, и потом константа.

Проверяю какая точка остановки у меня установлена (рис. [13](#fig:013)).

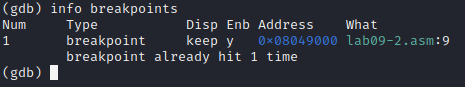


Figure 13: Точки останова

Определяю адрес предпоследней строки программы из рисунка 12 и устанавливаю новую точку останова (рис. [14](#fig:014)).

Figure 14: Вторая точка останова

Figure 14: Вторая точка останова

Выполняю 5 инструкций с помощью команды stepi и прослеживаю за изменением значений регистров.

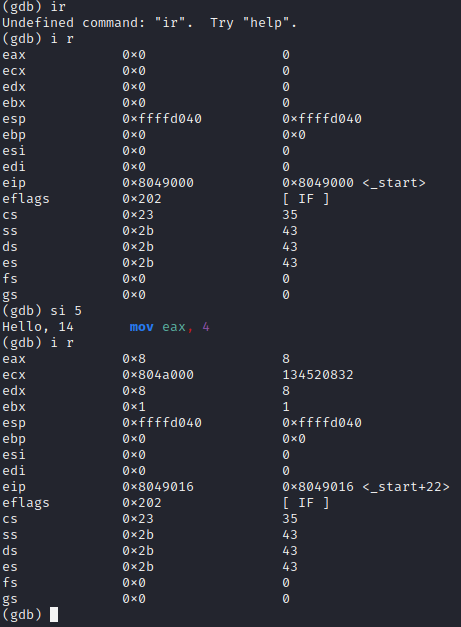


Figure 15: Выволнение 5 инструкций

Изменились значения регистров eax, ecx, edx, ebx, eip.

Смотрю значение переменной msg1 по имени (рис. [16](#fig:016)).

Figure 16: Значение переменной msg1

Figure 16: Значение переменной msg1

Смотрю значение переменной msg2 по адресу (рис. [17](#fig:017)).

Figure 17: Значение переменной msg2

Figure 17: Значение переменной msg2

Изменяю значение регистра msg1 с помощью команды set, задав ей в качестве аргумента имя регистра (рис. [18](#fig:018)).

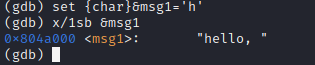


Figure 18: Новое значение регистра msg1

‘H’ поменялся на ‘h’.

Изменяю значение регистра msg2 с помощью команды set, задав ей в качестве аргумента имя регистра (рис. [19](#fig:019)).

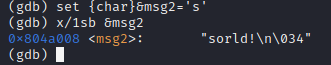


Figure 19: Новое значение регистра msg2

Вывожу значение регистра edx в разных форматах (рис. [20](#fig:020)).

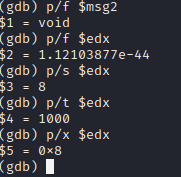


Figure 20: Значение регистра в разных форматах

Изменяю значение регистра ebx с помощью команды set (рис. [21](#fig:021)).

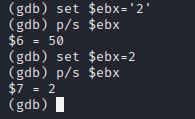


Figure 21: Изменение значения регистра командой set

Когда значение 2 (как целое число, без кавычек), gdb интерпретирует его как целое число и выводит его значение как 2, а когда с кавычками, то выводится номер соответствующий элементу в ASCII.

Завершаю выполнение программы с помощью команды continue (сокращенно c) и выхожу из GDB с помощью команды quit (сокращенно q) (рис. [22](#fig:022)).

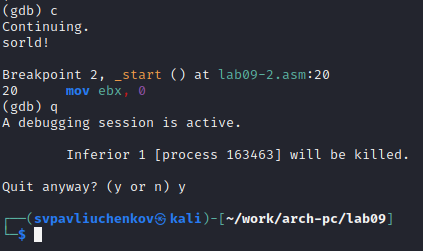


Figure 22: Завершение выполнения программы и выход из GDB

* Обработка аргументов командной строки в GDB Скопировал файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем lab09-3.asm, создал исполняемый файл и загрузил исполняемый файл в отладчик, указав аргументы (рис. [23](#fig:023)).

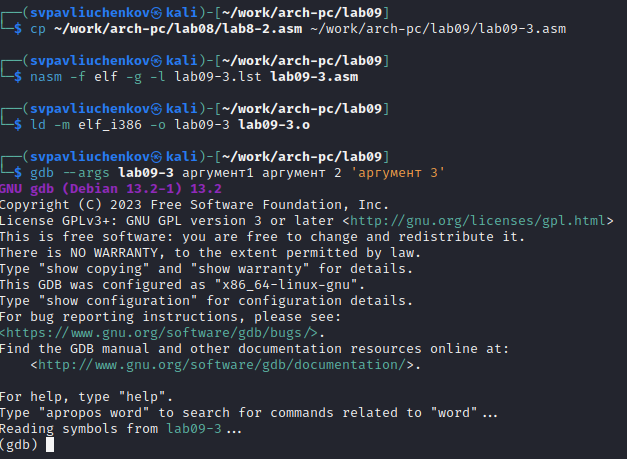


Figure 23: Загрузка файла с аргументами в отладчик

Устанавливаю брейкпоинт и запускаю отладчик (рис. [24](#fig:024)).

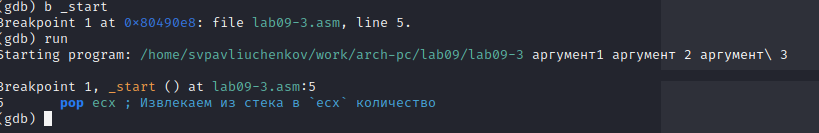


Figure 24: Запуск отладчика

Смотрю остальные позиции стека – по адресу [esp+4] располагается адрес в памяти где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д(рис. [25](#fig:025)).

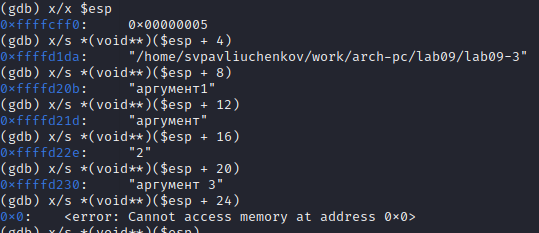


Figure 25: Запуск отладчика

Шаг изменения адреса равен 4 потому, что в архитектуре x86 данные обычно выравниваются по границе 4 байта.

* Задание для самостоятельной работы Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму(рис. [26](#fig:026)).

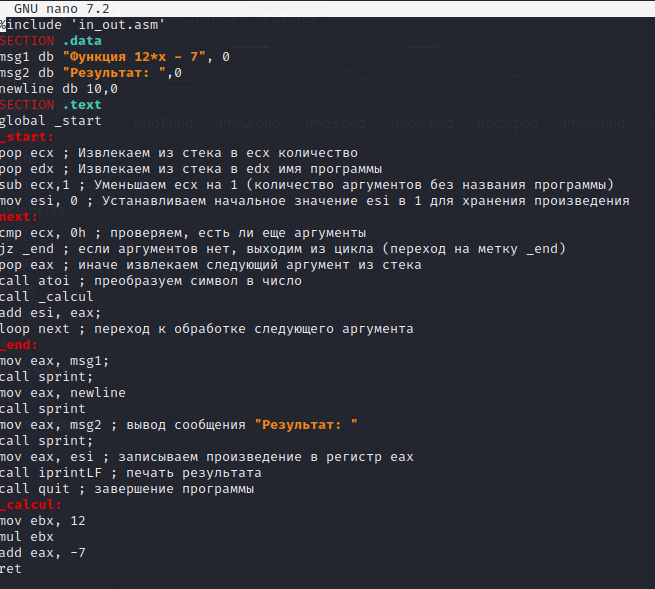


Figure 26: Преобразованная программа

Запуск программы и проверки, что она работает правильно(рис. [27](#fig:027)).

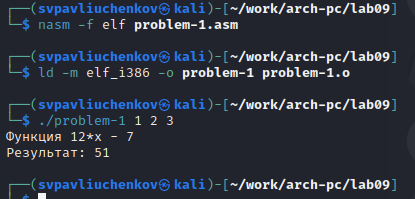


Figure 27: Создание и запуск исполнительного файла

Все работает правильно.

Ввожу в файл problem-2.asm текст программы из листинга 9.3 (рис. [28](#fig:028)).

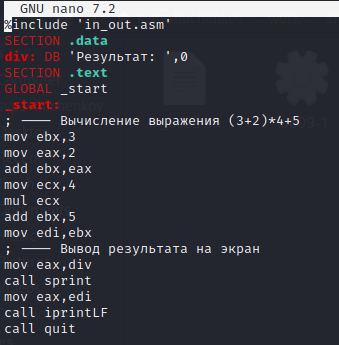


Figure 28: Программа из листинга 9.3

Запуск программы и проверка, что она работает некорректно (рис. [29](#fig:029)).

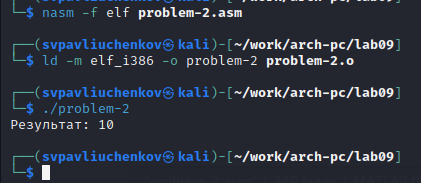


Figure 29: Создание и запуск исполнительного файла

Отладка показа, что сначала в ebx записывается 3, потом в eax 2, после в ebx записывается 5 (сумма прошлых двух). после ecx становится 4 и eax становится 8 (2 \* 4), и наконец ebx = 10(рис. [30](#fig:030)). Ошибка заключается в том, что умножение происходит на eax, а не на ebx.

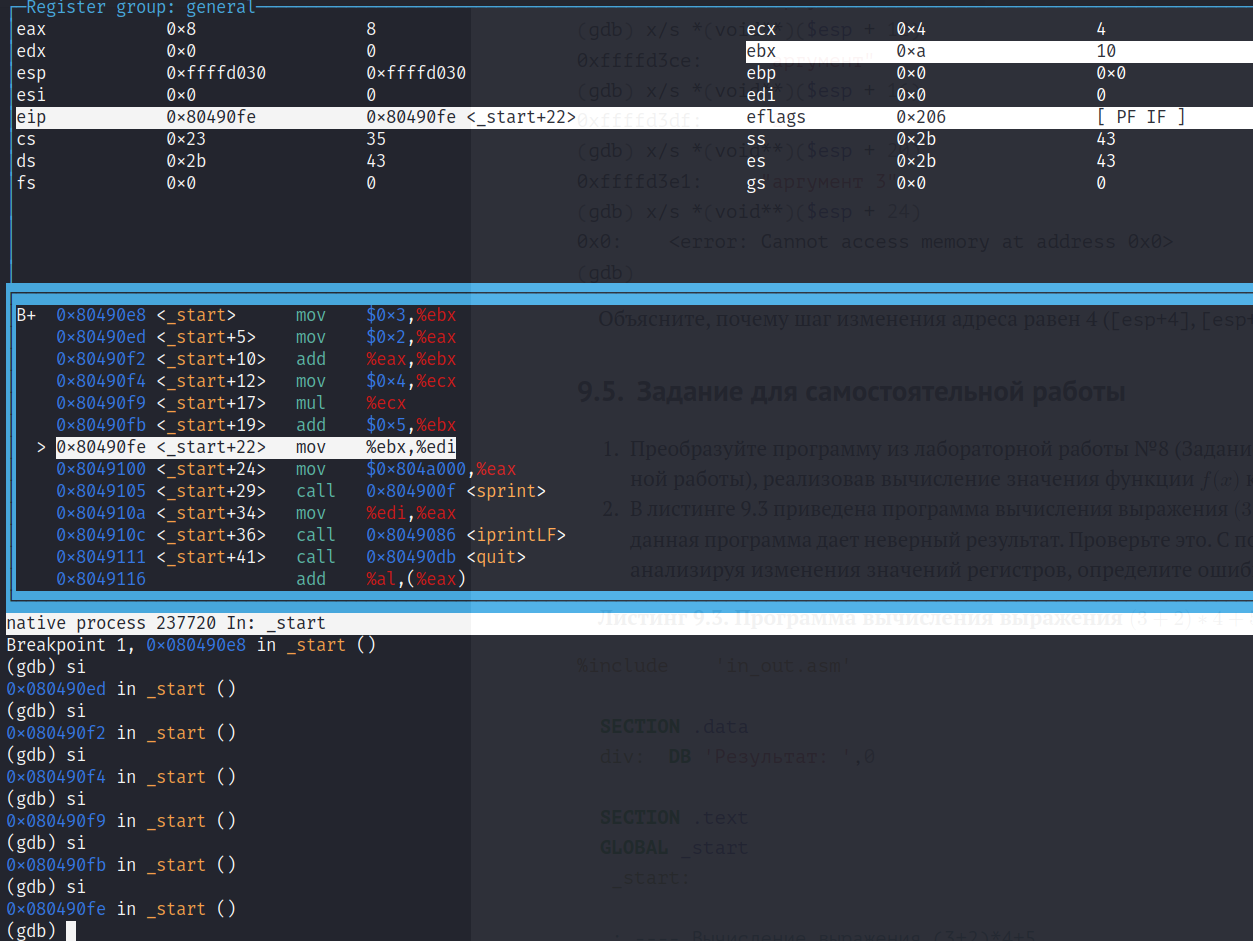


Figure 30: Отладка

Меняю умножение на ebx в программе. (рис. [31](#fig:031))

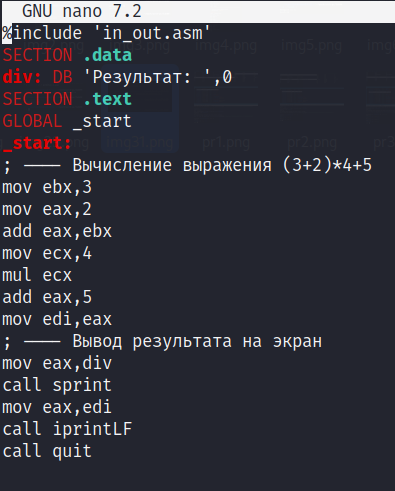


Figure 31: Исправленная программа

Запускаю программу, чтобы проверить, что все верно. (рис. [32](#fig:032))

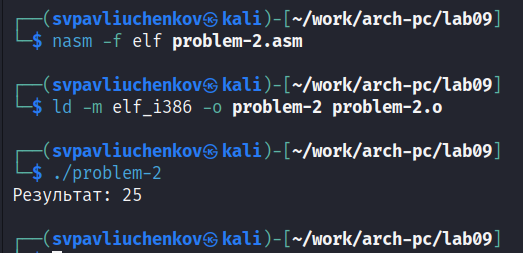


Figure 32: Создание и запуск исполнительного файла

В этот раз программа выводит верный ответ - 25.

# 4 Выводы

Выполнив эту работу я научился писать и использовать подпрограммы и ознакомился с отладчиком GDB, и его инструментами и функциями.

# Список литературы

::: Лабораторная работа №9 ::: GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/ ::: NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/ :::