Отчёт по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Юсупова Ксения Равилевна

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

## 2.1 Реализация подпрограмм в NASM

Создаем каталог для программам лабораторной работы № 9, переходим в него и со- здаём файл lab9-1.asm(рис. 1).

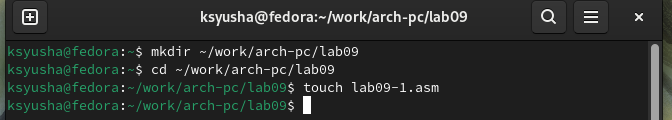


Рис. 1: Создали каталог с помощью команды mkdir и файл lab9-1.asm с помощью команды touch

Вводим в файл lab7-1.asm текст программы из листинга 9.1. (рис. 2).

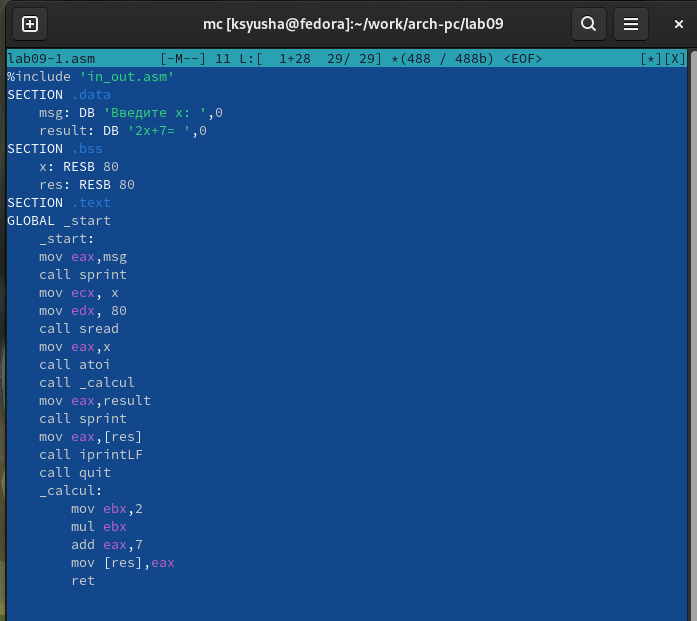


Рис. 2: Заполнили файл по листингу

Создаем исполняемый файл и запускаем его(рис. 3).

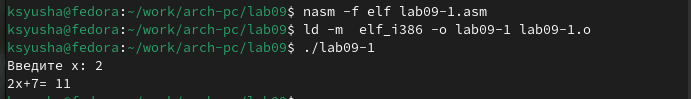


Рис. 3: Создали исполняемый файл и запустили его

Далее изменяем текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul.(рис. 4).

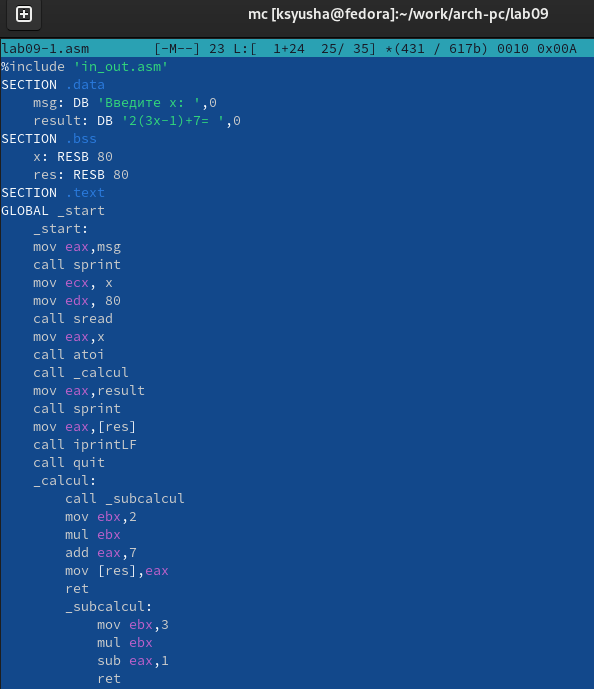


Рис. 4: Изменили файл

Создаем исполняемый файл и запускаем его((рис. 5).

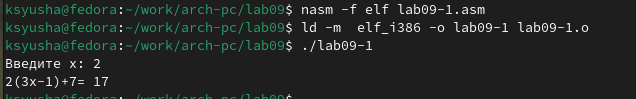


Рис. 5: Создали исполняемый файл и запустили его

## 2.2 Отладка программам с помощью GDB

Создаем исполняемый новый файл lab09-2.asm в каталоге ~/work/arch-pc/lab09.с помощью команды touch(рис. 6).

Создали файл lab09-2.asm

Рис. 6: Создали файл lab09-2.asm

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом(рис. 7).

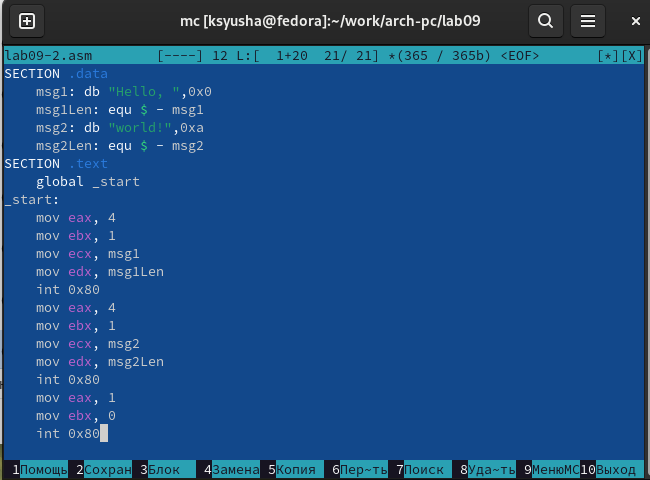


Рис. 7: Заполнили файл

Получаем исходный файл с использованием отладчика gdb(рис. 8).

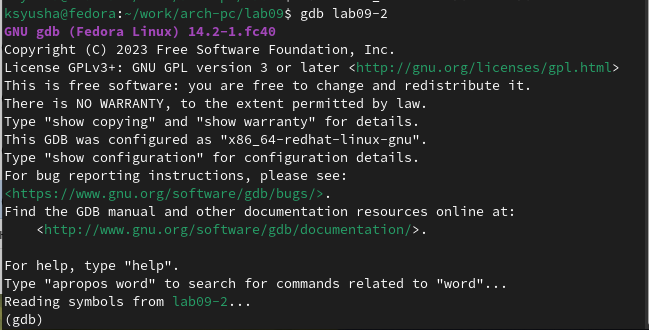


Рис. 8: Загрузили исходный файл в отладчик

Проверяем работу программы, запустив ее в оболочке GDB с помощью команды run.(рис. 9)

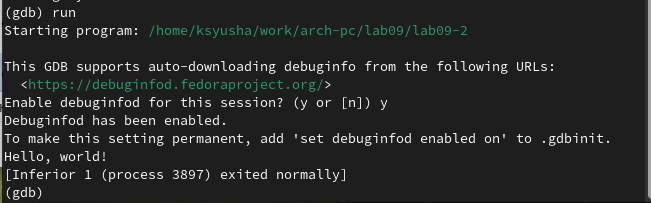


Рис. 9: Запустили программу командой run

Для более подробного анализа программы установим брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустим её(рис. 10).



Рис. 10: Запустили программу с брейкпоином

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. 11).



Рис. 11: Просмотрели дисассимилированный код программы

Переключаемся на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel(рис. 12).

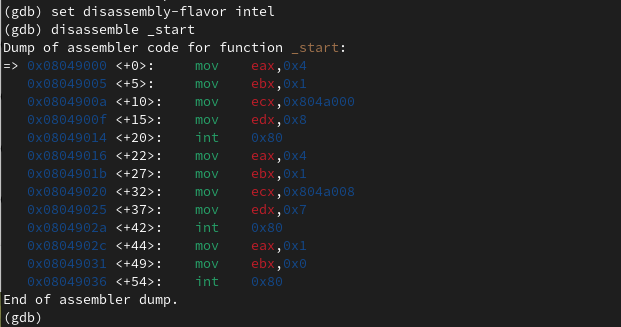


Рис. 12: Переключаемся на синтаксис Intel

Главное различие между ATT и Intel синтаксисом ассемблера — порядок операндов: ATT использует источник, назначение, а Intel — назначение, источник. ATT использует префикс % для регистров (например, %eax), а Intel — нет (например, eax). Формат мемориальных адресов также немного отличается. Многие ассемблеры поддерживают оба синтаксиса.

Включим режим псевдографики для более удобного анализа программы(рис. 13).

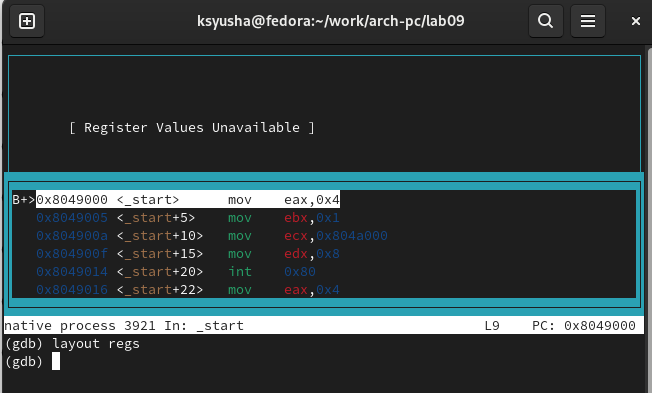


Рис. 13: Включили режим псевдографики

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверим это с помощью команды info breakpoints. Установим еще одну точку останова по адресу инструкции.(рис. 14).

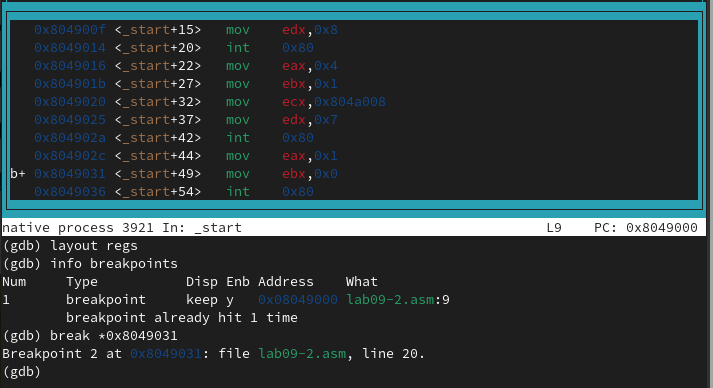


Рис. 14: Использовали команду info breakpoints и создали новую точку останова

Посмотрим информацию о всех установленных точках останова(рис. 15).

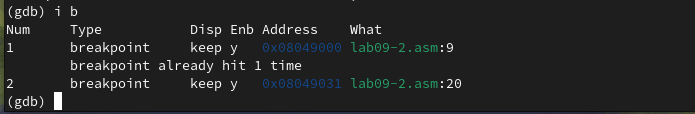


Рис. 15: Посмотрели информацию

Выполняем 5 инструкций командой si(рис. 16).

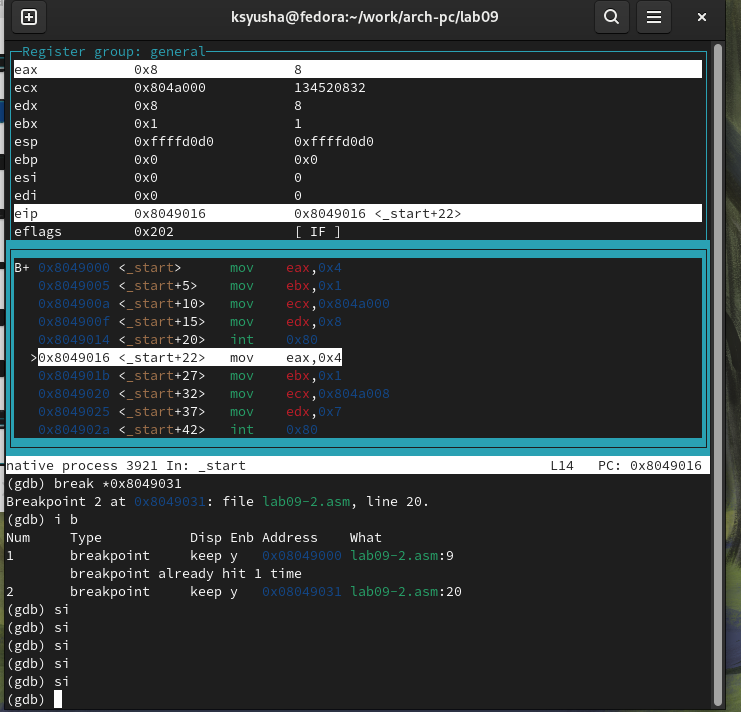


Рис. 16: Отследили регистры

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

Смотрим значение переменной msg1 по имени(рис. 17).

Посмотрели значение переменной

Рис. 17: Посмотрели значение переменной

Смотрим значение переменной msg2 по адресу(рис. 18).

Посмотрели значение переменной

Рис. 18: Посмотрели значение переменной

Изменим первый символ переменной msg1.(рис. 19)

Изенили символ

Рис. 19: Изенили символ

Изменим первый символ переменной msg2(рис. 20).

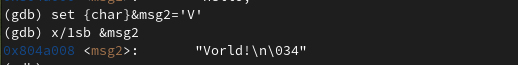


Рис. 20: Изенили символ

Смотрим значение регистра edx в разных форматах(рис. 21).

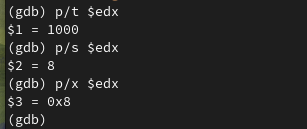


Рис. 21: Изенили символ

Изменяем регистор ebx(рис. 22).

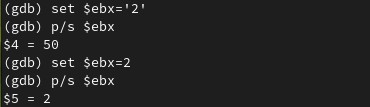


Рис. 22: Изменили регистор

Вывод команд p/s $ebx имеет разные значения, так как команда без кавычек присваивает регистру вводимое значение.

Завершаем выполнение программы с помощью команды continue (сокращенно c) и выходим из GDB с помощью команды quit (сокращенно q).(рис. 23)

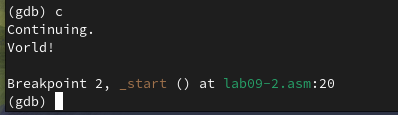


Рис. 23: Прописали команды c и quit

Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm.(рис. 24).

Скопировали файл

Рис. 24: Скопировали файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. 25)

Создали и запустили в отладчике файл

Рис. 25: Создали и запустили в отладчике файл

Для начала установим точку останова перед первой инструкцией в программе и запустим ее.(рис. 26).

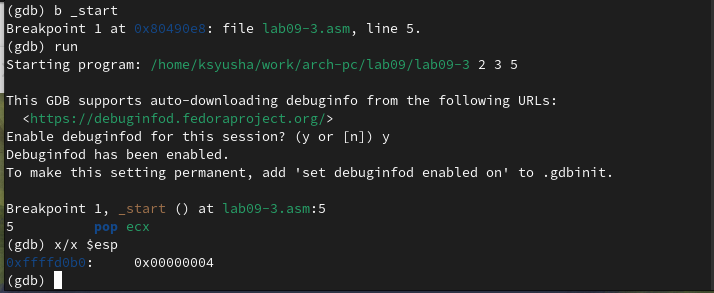


Рис. 26: Устанавливаем точку

Смотрим позиции стека по разным адресам. (рис. 27).



Рис. 27: Изучили полученныу данные

Шаг изменения адреса равен 4, потому что адресные регистры имеют размер- ность 32 бита или же 4 байта.

## 2.3 Задание для самостоятельной работы

### 2.3.1 Задание 1

1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму.

Создаем исполняемый новый файл lab09-4.asm в каталоге ~/work/arch-pc/lab09.с помощью команды touch и скопировали в него файл lab8-4.asm(рис. 28).

Создали исполняемый файл и скопировали в него lab8-4.asm

Рис. 28: Создали исполняемый файл и скопировали в него lab8-4.asm

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму(рис. 29).

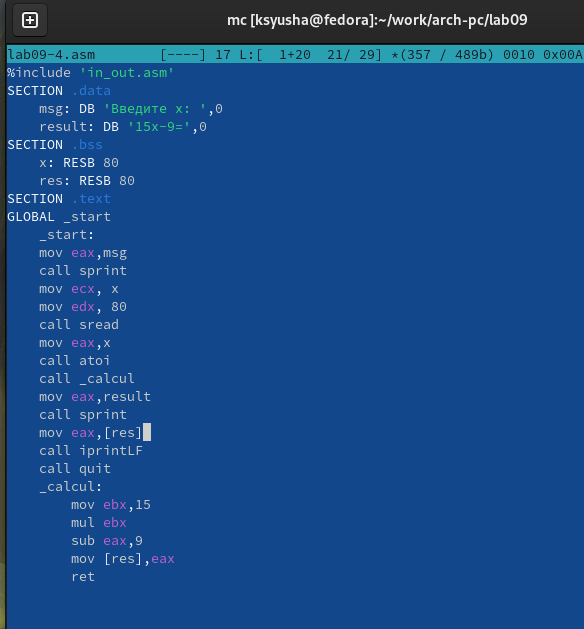


Рис. 29: Изменили файл

Создаем исполняемый файл и запускаем его((рис. 30).

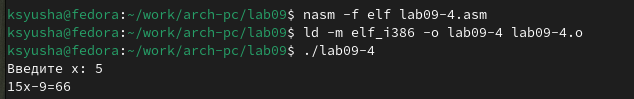


Рис. 30: Создали исполняемый файл и запустили его

### 2.3.2 Задание 2

1. В листинге 9.3 приведена программа вычисления выражения (3 + 2) ∗ 4 + 5. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

Создаём файл lab09-4.asm и заполняем его в соответствии с листингом 9.3 (рис. 31).

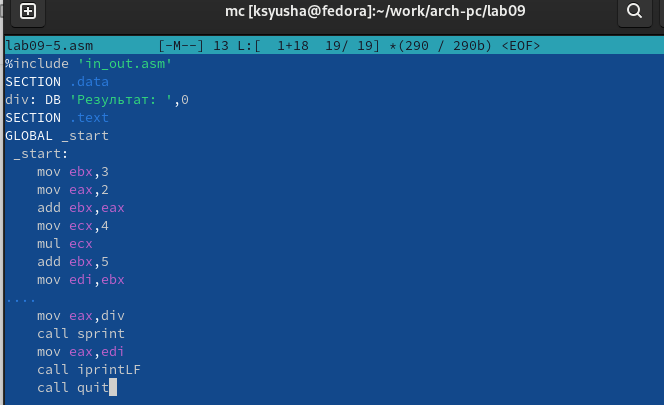


Рис. 31: Заполнили файл

Создаем исполняемый файл и запускаем его, проверяя вывод ошибочного значения(рис. 32).

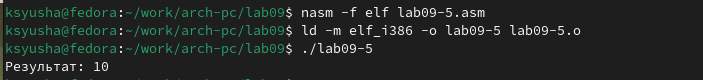


Рис. 32: Создали исполняемый файл и запустили его, заметили неправильный вывод

Загружаем исходный файл в отладчик(рис. 33).

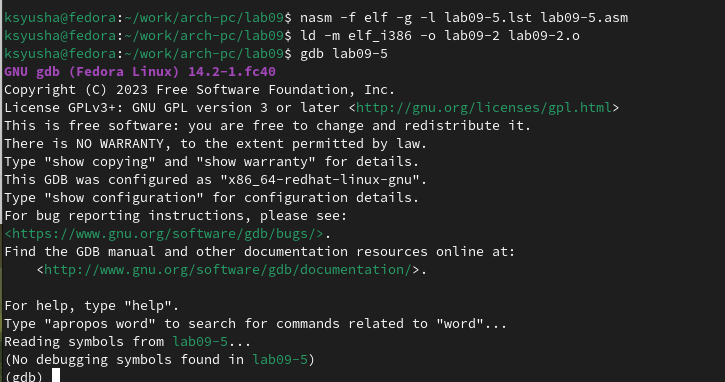


Рис. 33: Загрузили исходный файл в отладчик

Запускаем файл в отладчике GDB и смотрим на изменение регистров командой si(рис. 34)

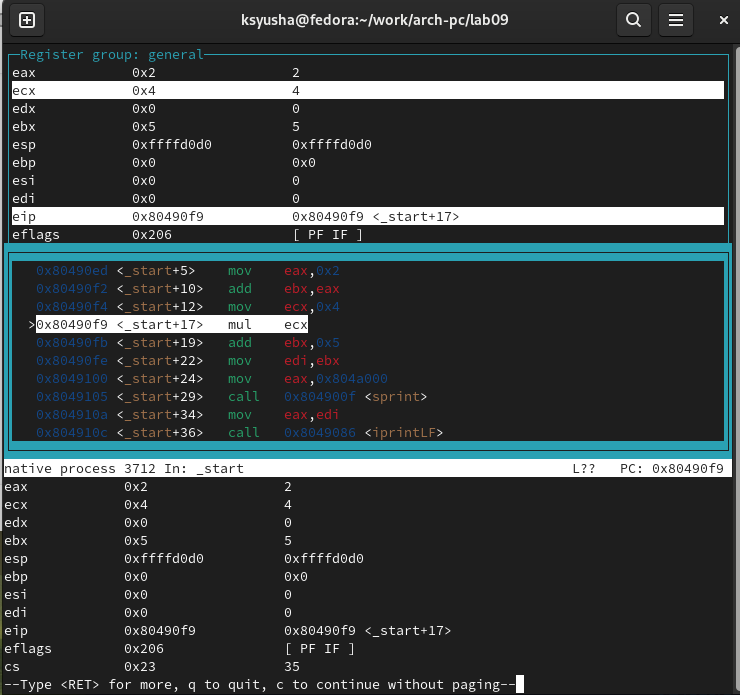


Рис. 34: Определяем ошибку

Изменяем программу так, чтобы вывод был верным 35).

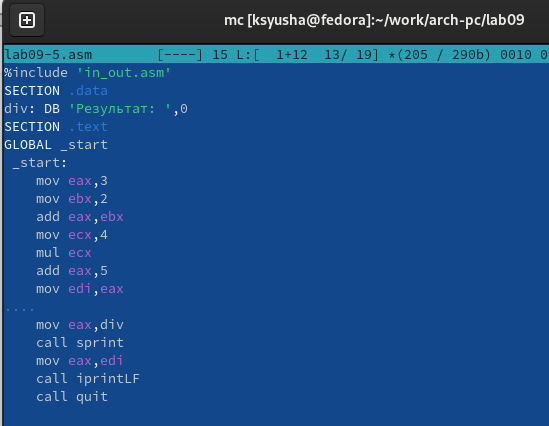


Рис. 35: Исправили ошибки в программе

Создаем исполняемый файл и проверяем корректность работы(рис. 36).

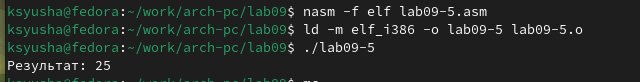


Рис. 36: Создали и запустили файл, убедились в верности вывода

# 3 Выводы

В ходе лабораторной работы мы приобретели навыки написания программ с использованием подпрограмм, и ознакомились с методами отладки при помощи GDB и его основными возможностями.

# Список литературы