Отчёт по лабораторной работе №9

Архитектура компьютера НММбд-03-24

Туева Анастасия Юрьевна

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. Выполнение лабораторной работы
2. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму.
3. 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.

# 3 Выполнение лабораторной работы

Создаём каталог для программ лабораторной работы №9, переходим в него и создаём файл lab09-1.asm. (рис. 1).

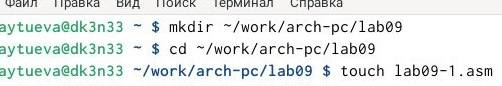


Рис. 1: Создание файла

В качестве примера рассмотрим программу вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы \_calcul. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Вводим в файл lab09-1.asm текст программы из листинга 9.1. (рис. 2).

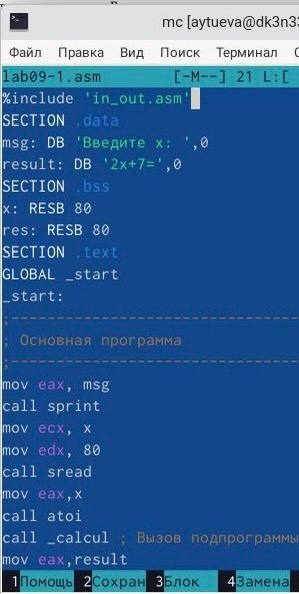


Рис. 2: Редактирование файла

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

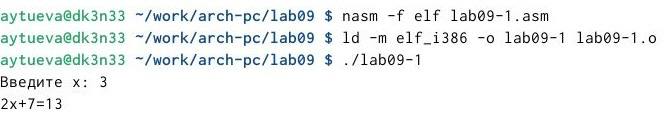


Рис. 3: Запуск исполняемого файла

Изменим текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x) = 2x + 7, g(x) = 3x - 1. (рис. 4).

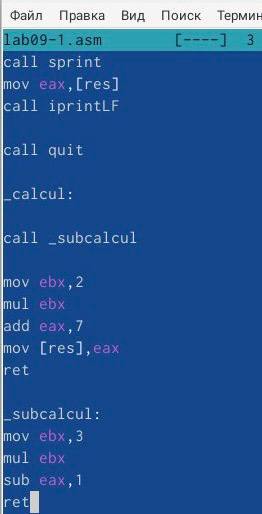


Рис. 4: Редактирование файла

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

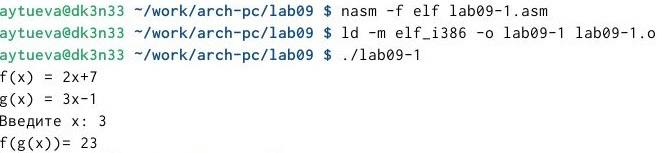


Рис. 5: Запуск исполняемого файла

Создаем файл lab09-2.asm. Вводим в него программу из листинга 9.2. (рис. 6).

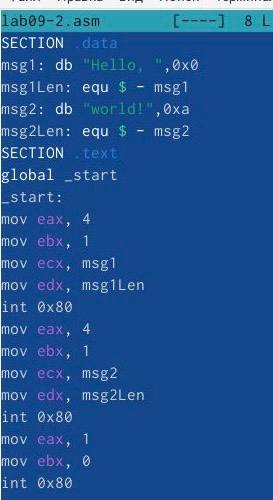


Рис. 6: Редактирование файла lab09-2.asm

Транслируем текст программы с ключом ‘-g’. Загружаем исполняемый файл в gdb (рис. 7).

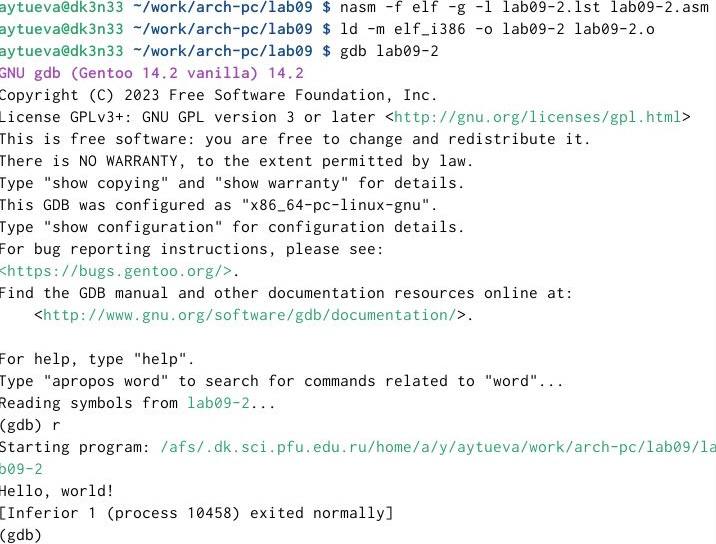


Рис. 7: Исполнение программы

Для более подробного анализа программы установим брейкпойнт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустим её. (рис. 8).

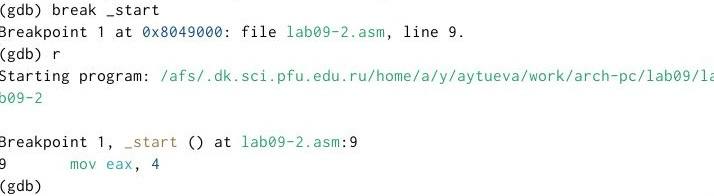


Рис. 8: Исполнение програмы брейкпойнт

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start. (рис. 9).



Рис. 9: Просмотр дисассимилированного кода программы

Переключимся на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel. Различия между синтаксисом ATT и Intel заключаются в порядке операндов(ATT - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (ATT - pазмер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом $; Intel - Размер операндов неявно определяется контекстом, как ax, eax, непосредственные операнды пишутся напрямую), именах регистров(ATT - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов). (рис. 10).

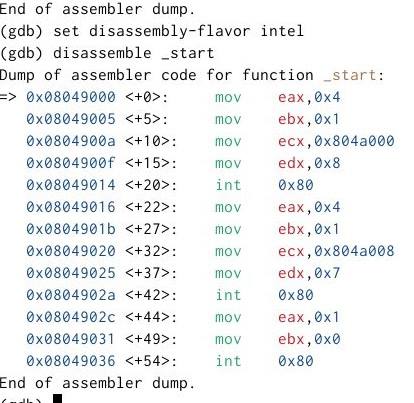


Рис. 10: Просмотр дисассимилированного кода программы с синтаксисом Intel

Включим режим псевдографики для более удобного анализа программы. (рис. 11).

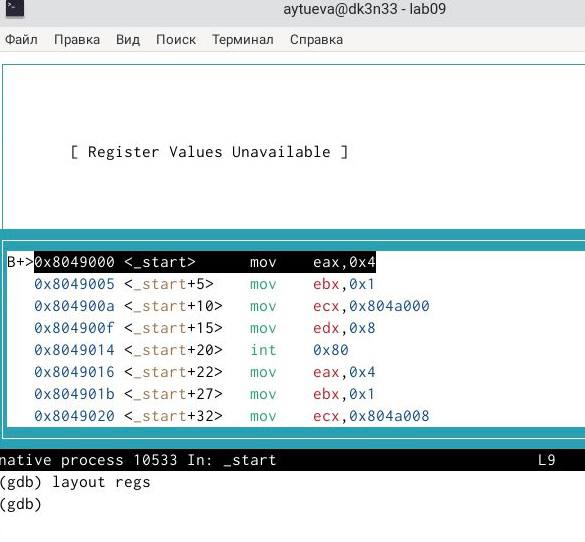


Рис. 11: Переход в режим псевдографики

Посмотрим наличие меток и добавим еще одну метку на предпоследнюю инструкцию. (рис. 12).

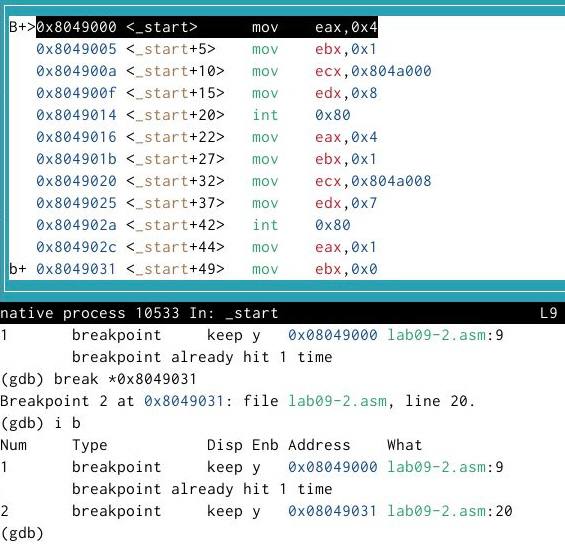


Рис. 12: Наличие меток

С помощью команды посмотрим значение переменной msg1. (рис. 13).

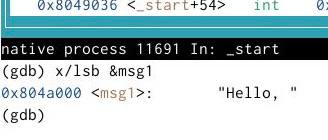


Рис. 13: Значение переменной msg1

Посмотрим значение второй переменной msg2. (рис. 14).

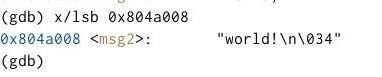


Рис. 14: Значение переменной msg2

С помощью команды set изменим значение переменной msg1. (рис. 15).

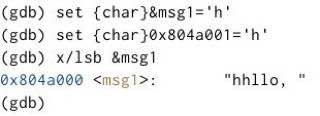


Рис. 15: Изменение значения переменной msg1

Изменим переменную msg2. (рис. 16).

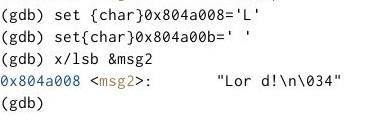


Рис. 16: Изменение значения переменной msg2

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.(рис. 17).

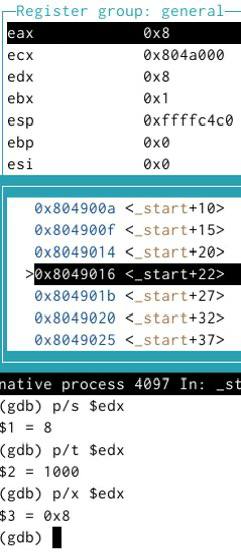


Рис. 17: Значение регистра edx

Скопируем файл lab8-2.asm в файл с именем lab09-3.asm. Создадим исполняемый файл. Загрузим исполняемый файл в отладчик, указав аргументы. (рис. 18).

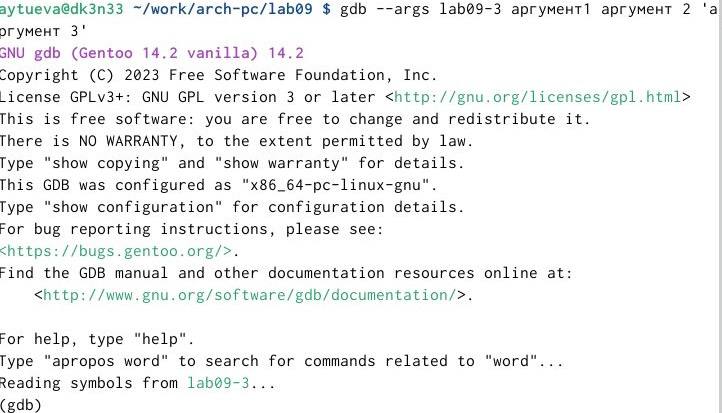


Рис. 18: Загрузка файла lab09-3.asm в отладчик

Поставим метку на \_start и запустим файл. (рис. 19).



Рис. 19: Запуск файла lab09-3 через метку

Проверим адрес вершины стека и убедимся, что там хранится 5 элементов. (рис. 20).

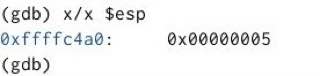


Рис. 20: Адрес вершины стека

Посмотрим остальные позиции стека – по адесу [esp+4] располагается адрес в памяти где находится имя программы, по адесу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д. Шаг изменения адреса равен 4 байтам, потому что мы работаем с 32-битной системой (x86), а указатели (void \*\*) в такой системе занимают 4 байта. Ошибка Cannot access memory at address 0x0 на $esp + 24 указывает на то, что закончились аргументы командной строки. (рис. 21).

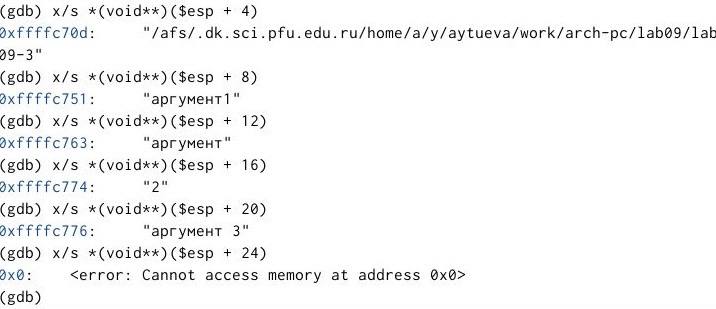


Рис. 21: Проверка остальных позиций стека

# 4 Выполнение самостоятельной работы

Преобразем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму.(рис. 22).

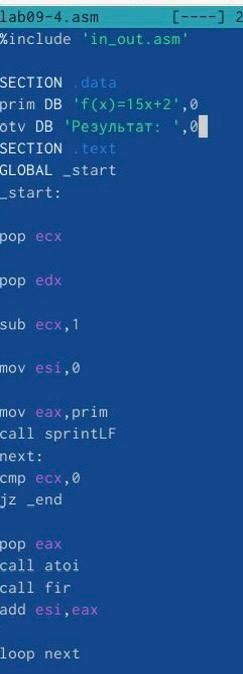


Рис. 22: Текст программы lab09-4.asm

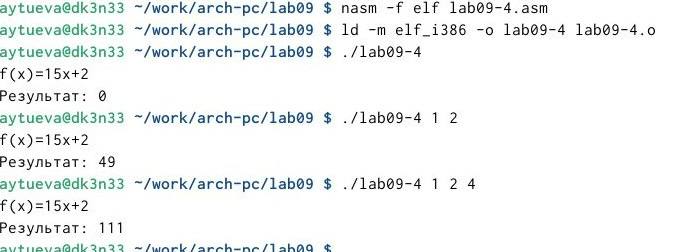


Рис. 23: Запуск программы

Перепишем программу и попробуем запустить ее, чтобы увидеть ошибку. Ошибка арифметическая, так как вместо 25,программа выводит 10. (рис. 24).

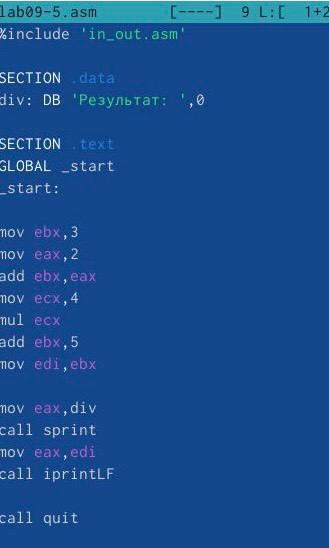


Рис. 24: Текст программы

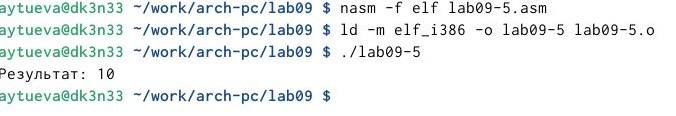


Рис. 25: Запуск программы

После появления ошибки, я запустила программу в отладчике. (рис. 26).

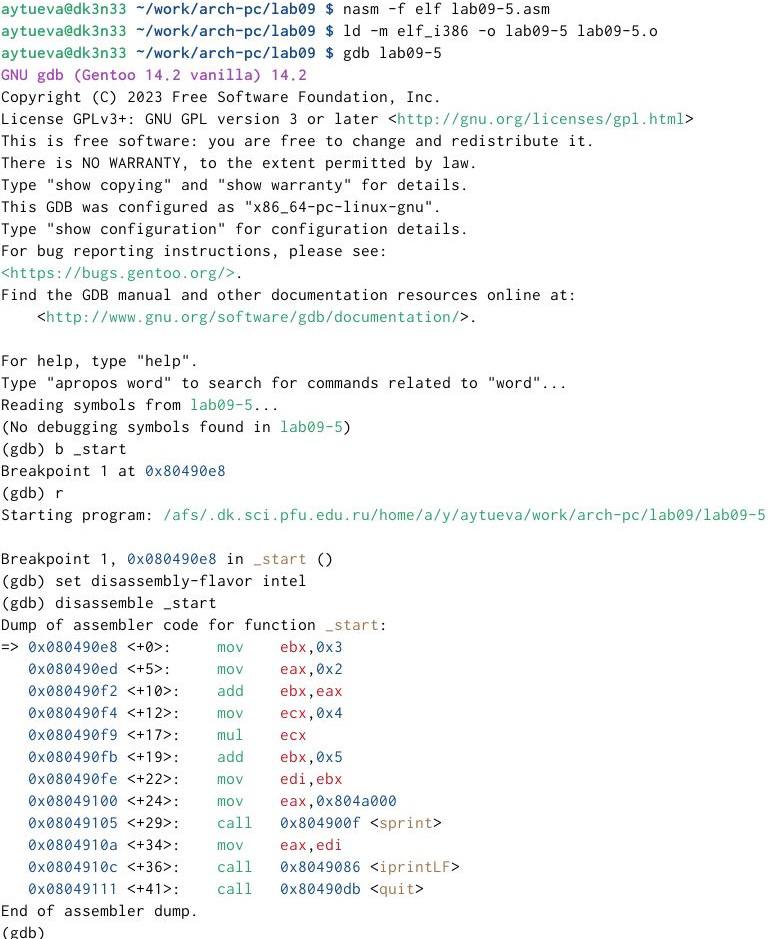


Рис. 26: Запуск файла lab09-3 через метку

Откроем регистры и проанализируем их. Некоторые регистры стоят не на своих местах. Исправим это. Изменим регистры и запустим программу. Программа вывела ответ 25, то есть все работает правильно.(рис. 27).

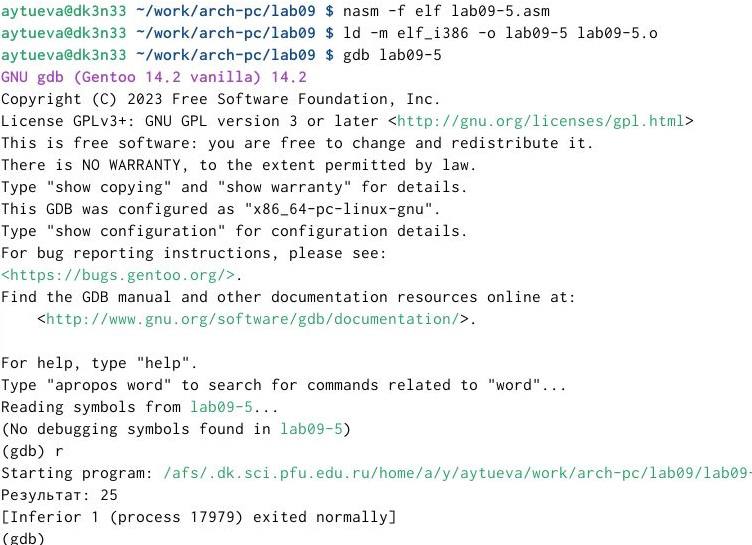


Рис. 27: Повторный запуск программы

# 5 Выводы

В результате выполнения лабораторной работы я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки в NASM.