Отчёт по лабораторной работе №9

Архитектура компьютера НММбд-03-24

Топорова Дарья Сергеевна

Содержание

# 1 Цель работы

Целью работы является приобретение навыков написания программ с использованием подпрограмм, а также знакомство с методами отладки при помощи GDB.

# 2 Задание

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

1. Создаём каталог для программ лабораторной работы №9, перехожу в него и создаю файл lab09-1.asm.



Рис. 1: Создание каталог и файла

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

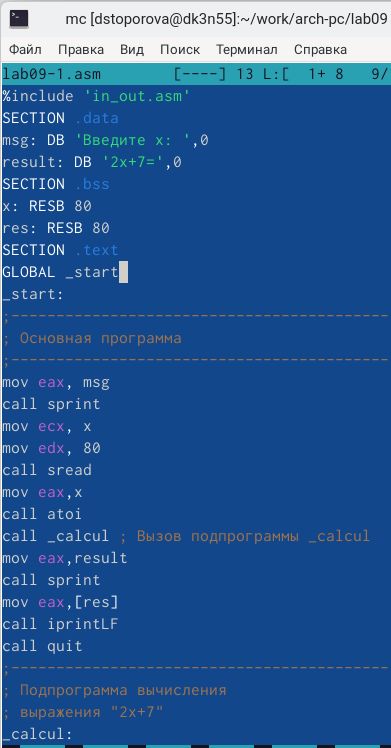


Рис. 2: программа для вычисления арифметического выражения 𝑓(𝑥) = 2𝑥 + 7

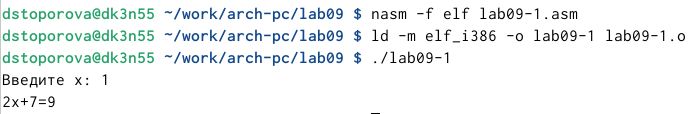


Рис. 3: Исполнение программы из листинга 9.1

1. 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. Запустим исправленную программу. Число проходов цикла не соответствует значению, введенному с клавиатуры.

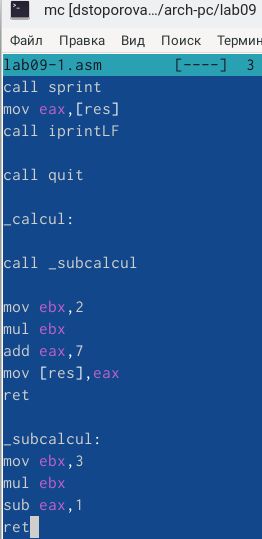


Рис. 4: Исправленный текст программы lab09-1.asm

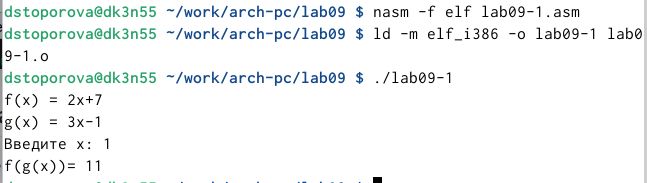


Рис. 5: Исполнение программы lab09-1

1. Создаем файл lab09-2.asm. Вводим в него программу из листинга 9.2. Транслируем текст программы с ключом ‘-g’. Загружаем исполняемый файл в gdb.

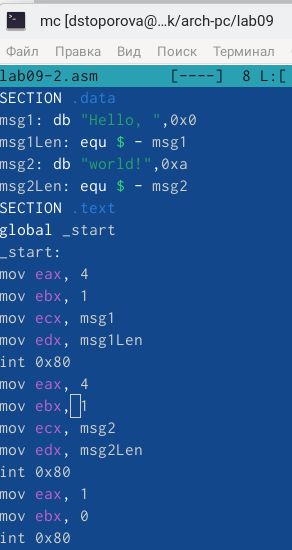


Рис. 6: Текст программы из листинга 9.2

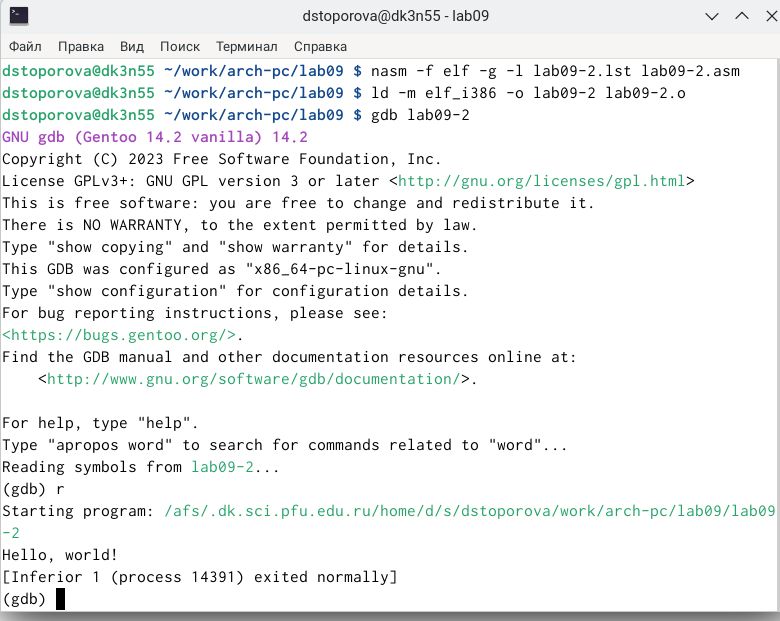


Рис. 7: Исполнение программы

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



Рис. 8: Исполнение программы с брейкпоинт

1. Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start.

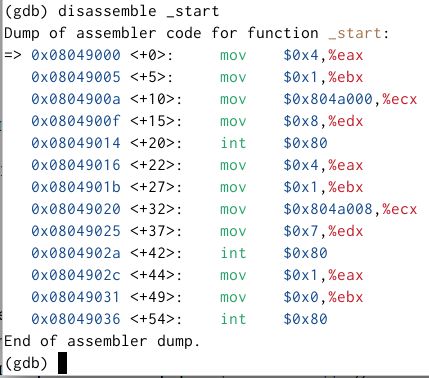


Рис. 9: Просмотр дисассимилированного кода программы

1. 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: Просмотр дисассимилированного кода программы с синтаксисом Intel |

Рис. 10: Просмотр дисассимилированного кода программы с синтаксисом Intel

1. Включим режим псевдографики для более удобного анализа программы.

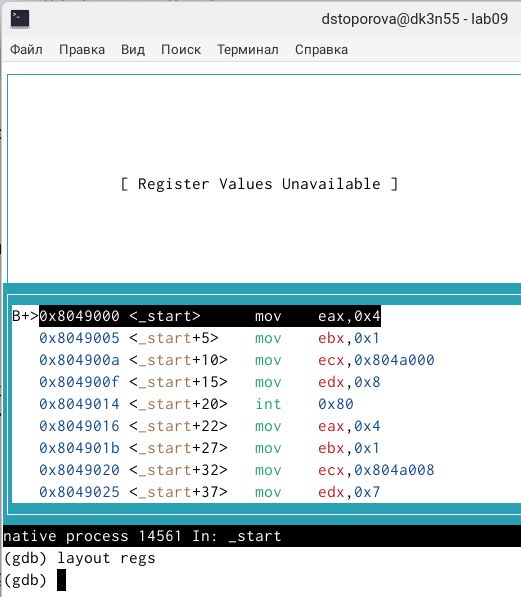


Рис. 11: Переход в режим псевдографики

1. Проверим установку точки останова на метке ’\_start’.

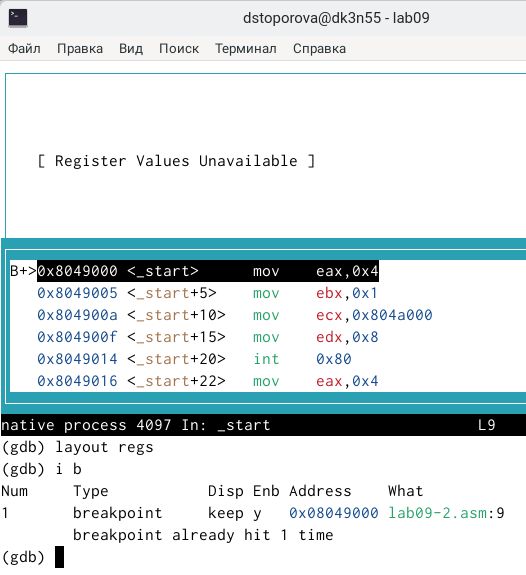


Рис. 12: Проверка установки точки останова

1. Установим еще одну точку останова по адресу инструкции. Определим адрес предпоследней инструкции (mov ebx,0x0) и установим точку останова.

|  |
| --- |
| Рис. 13: Установка новой точки останова |

Рис. 13: Установка новой точки останова

1. Просмотрим значение переменной msg1 по имени.



Рис. 14: Просмотр значения переменной

1. Просмотрим значение переменной msg2 по адресу.

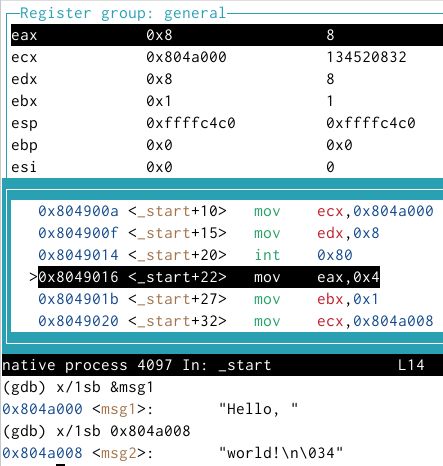


Рис. 15: Просмотр значения переменной

1. Изменим первый символ переменной msg1. Заменим любой символ во второй переменной msg2.

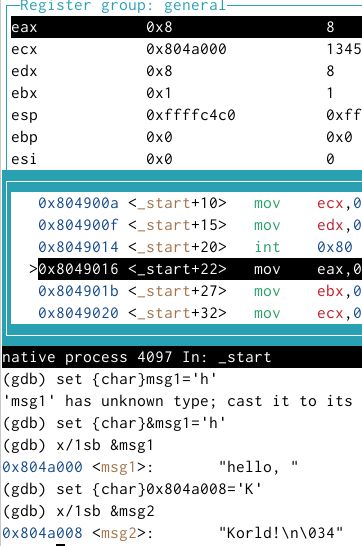
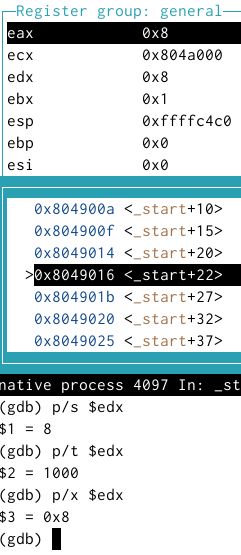


Рис. 16: Изменение переменной

1. Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx 
2. Скопируем файл lab8-2.asm в файл с именем lab09-3.asm. Создадим исполняемый файл. Загрузим исполняемый файл в отладчик, указав аргументы.

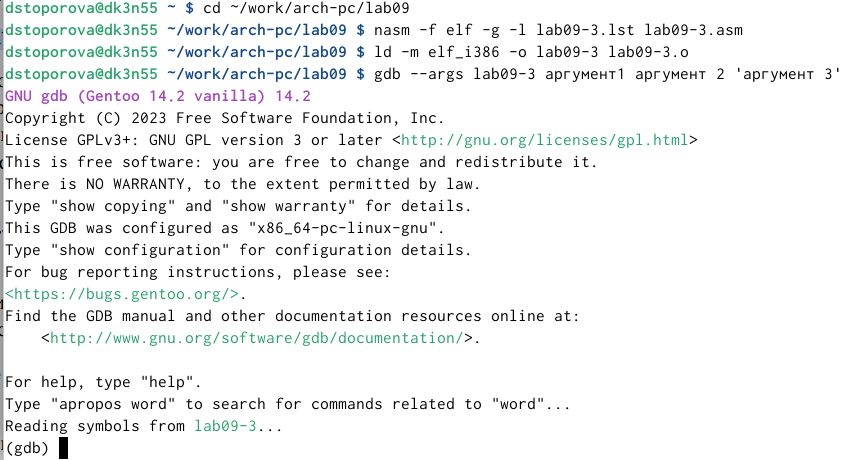


Рис. 17: Загрузка файла lab09-3.asm в отладчик

1. Для начала установим точку останова перед первой инструкцией в программе и запустим ее. Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

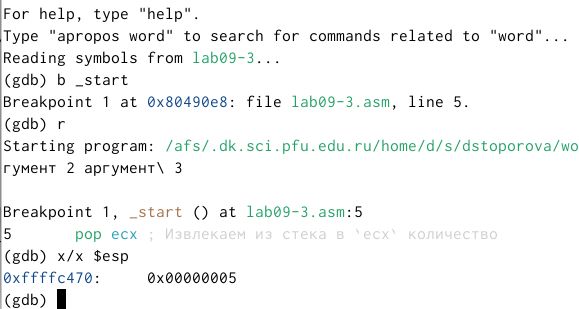


Рис. 18: Проверка стека

1. 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.

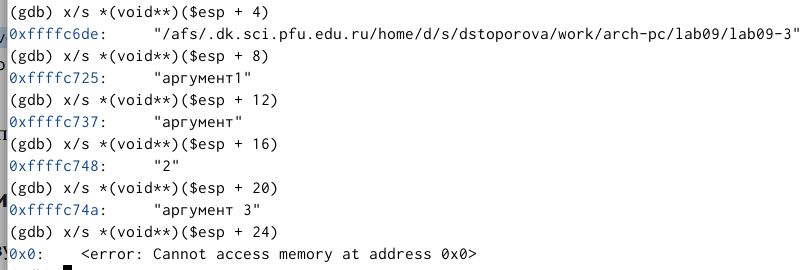


Рис. 19: Проверка остальных позиций стека

#Самостоятельная работа

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

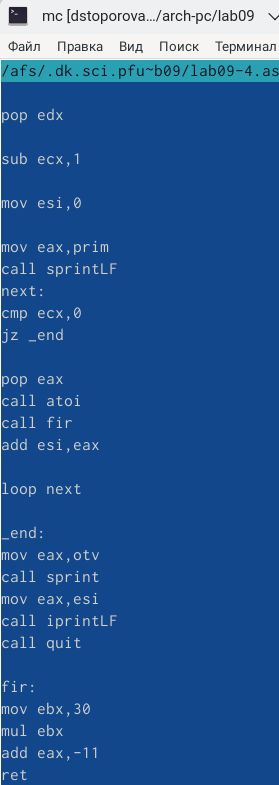


Рис. 20: Текст программы lab09-4.asm

1. В листинге 9.3 приведена программа вычисления выражения (3 + 2) ∗ 4 + 5. При запуске данная программа дает неверный результат. Ошибка в программе заключается в том, что инструкция mul ecx умножает значение в регистре eax на ecx, а результат записывает в eax. В исправленном варианте мы используем ebx для хранения промежуточного результата суммы, mul ecx умножает ebx на ecx, результат сохраняется в eax. Затем к результату в eax добавляется 5. Финальный результат сохраняется в edi и выводится на экран.

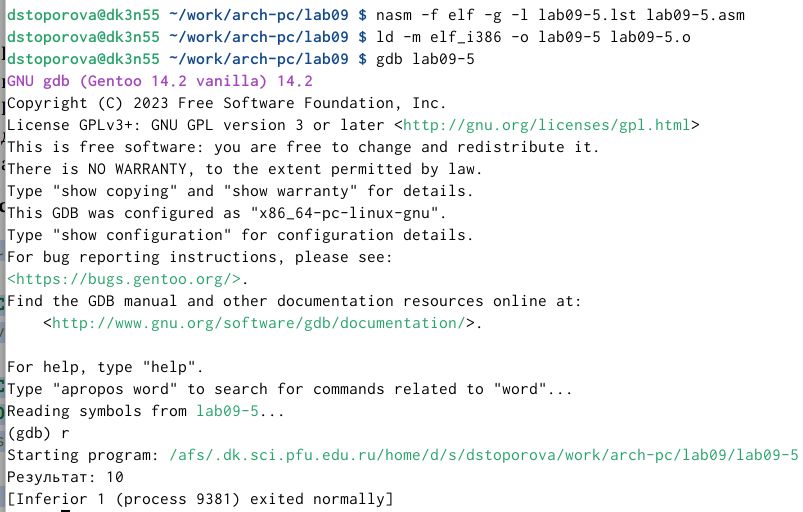


Рис. 21: Запуск программы

# 4 Выводы

Я приобрела навыки написания программ с использованием подпрограмм, а также знакомство с методами отладки при помощи GDB и его основными возможностями.