Шаблон отчёта по лабораторной работе

Простейший вариант

Дмитрий Сергеевич Кулябов

Содержание

# 1 *1 Цель работы*

Целью работы является приобретение навыков написания программ с использованием подпрограмм и ознакомление с методами отладки при помощи GDB и его основными возможностями.

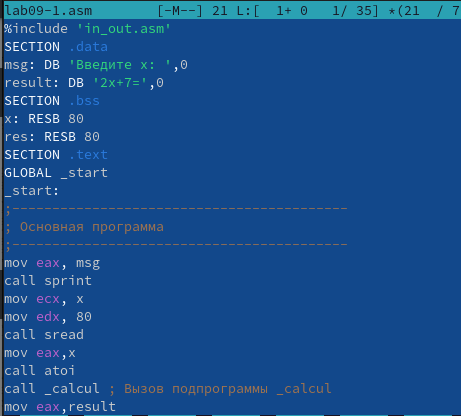
# 2 *2 Задание*

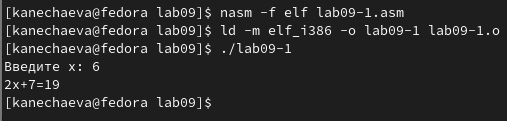
1. Реализация подпрограмм в NASM  
2. Отладка программ с помощью GDB  
 2.1. Добавление точек останова  
 2.2. Работа с данными программы в GDB  
 2.3. Обработка аргументов командной строки в GDB  
3. Задание для самостоятельной работы

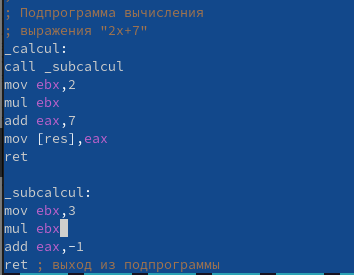
# 3 *3 Выолнение лабораторной работы*

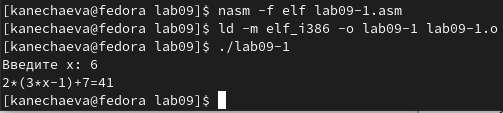
## 3.1 *1. Реализация подпрограмм в NASM*

Для начала я создаю каталог для программ лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm. (рис. [??]) Рис. 1. Создание рабочего пространства

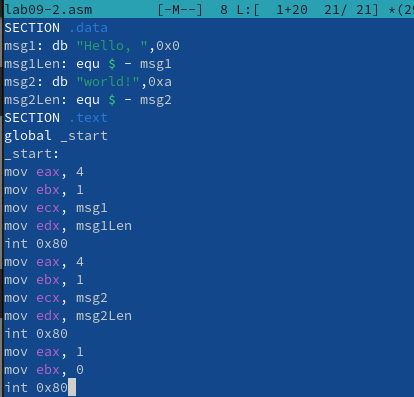
Ввожу в файл lab09-1.asm текст программы из листинга 9.1, скопировав заранее файл in\_out.asm в папку lab09. (рис. [??]) 

Создаю исполняемый файл и проверяю его работу. (рис. [??]) 

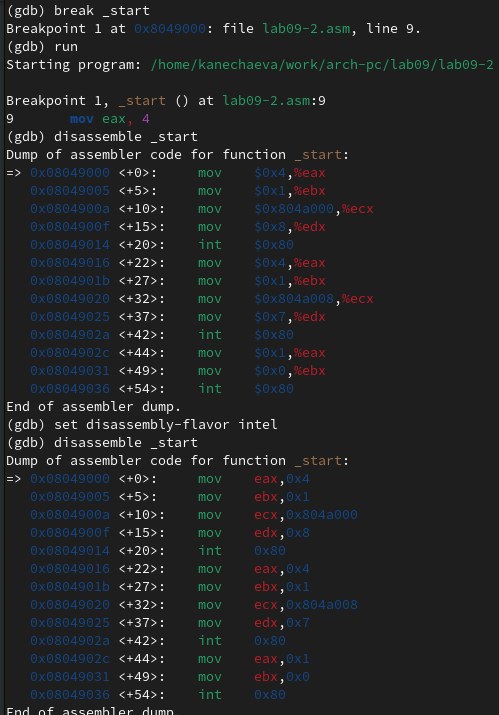
Теперь изменяю текст программы, добавив добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)). ((рис. [??]) 

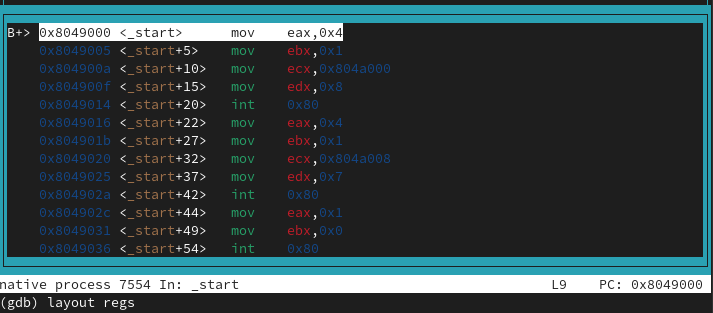
Создаю исполняемый файл и проверяю его работу. (рис. [??]) 

## 3.2 *2. Отладка программ с помощью GDB*

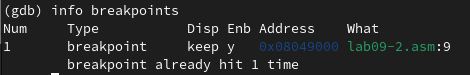
Создаю файл lab09-2.asm в каталоге ~/work/arch-pc/lab09 и ввожу в него текст программы из листинга 9.2. ((рис. [??]) 

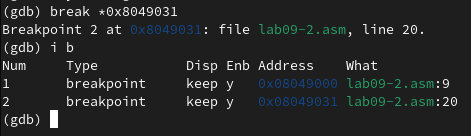
Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run. (рис. [??]) 

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаю её. Затем смотрю дисассимилированный код программы с помощью команды disassemble начиная с метки \_start. Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel. (рис. [??]) 

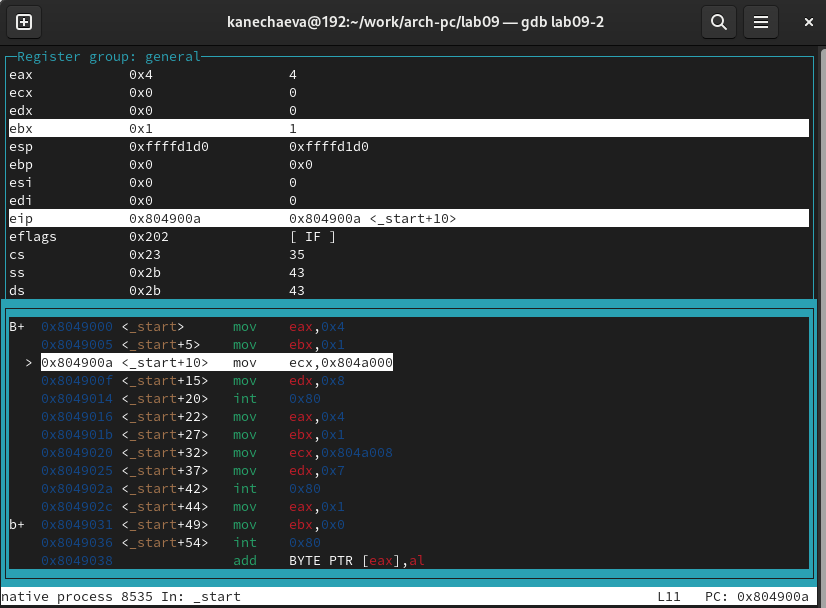
Включаю режим псевдографики для более удобного анализа программы. (рис. [??]) 

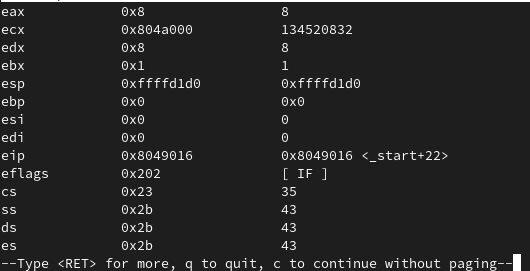
### 3.2.1 *2. 1. Добавление точек останова*

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяю это с помощью команды info breakpoints. (рис. [??]) 

Определяю адрес предпоследней инструкции (mov ebx,0x0) и устанавливаю точку останова.(рис. [??]) 

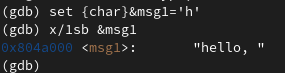
### 3.2.2 *2. 2. Работа с данными программы в GDB*

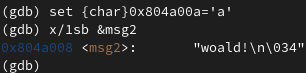
Выполняю 5 инструкций с помощью команды stepi (или si). (рис. [??])  Изуменяются значения регистров eax, ebx, ecx, edx.

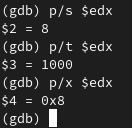
Смотрю содержимое регистров с помощью команды info registers. (рис. [??]) 

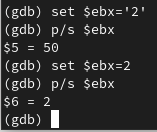
Теперь мне нужно значение переменной msg1 по имени.([??]) Рис. 14. Нахождение значения переменной по имени

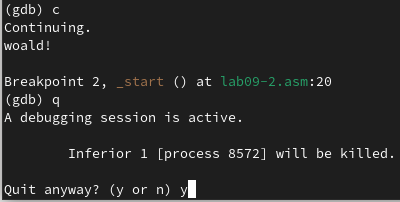
Затем я смотрю значение переменной msg2 по адресу. (рис. [??]) Рис. 15. Нахождение значения переменной по адресу

Изменяю первый символ переменной msg1. (рис. [??]) 

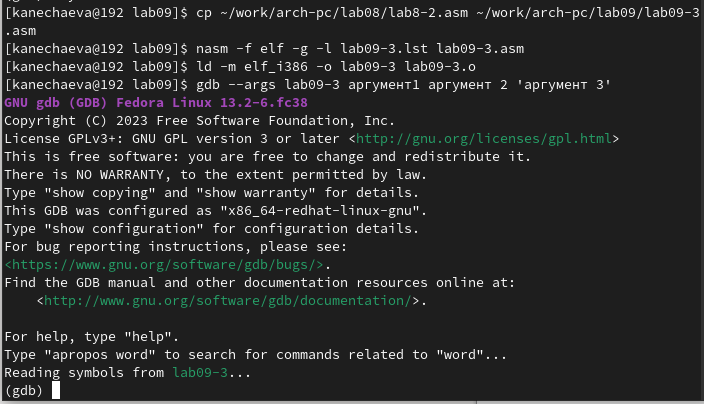
Заменяю третий символ во второй переменной msg2. (рис. [??]) 

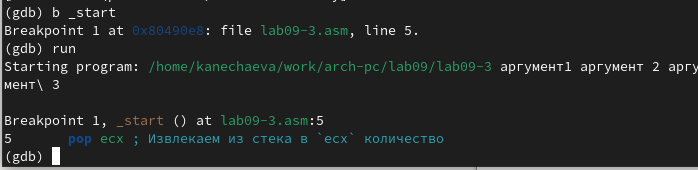
Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. (рис. [??]) 

С помощью команды set изменяю значение регистра ebx. (рис. [??])  Разница вывода команд p/s $ebx в том, что в первый раз программа выдает 50, т.к. это значение символа 2, а во второй раз уже сам символ 2.

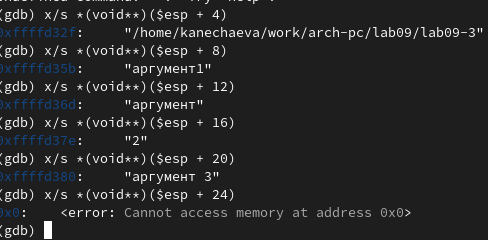
Завершаю выполнение программы с помощью команды continue (сокращенно c) и выхожу из GDB с помощью команды quit. (рис. [??]) 

### 3.2.3 *2. 3. Обработка аргументов командной строки в GDB*

Копирую 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. После этого создаю исполняемый файл и для загрузки в gdb программы с аргументами использую ключ –args. Теперь загружаю исполняемый файл в отладчик, указав определенные аргументы. (рис. [??]) 

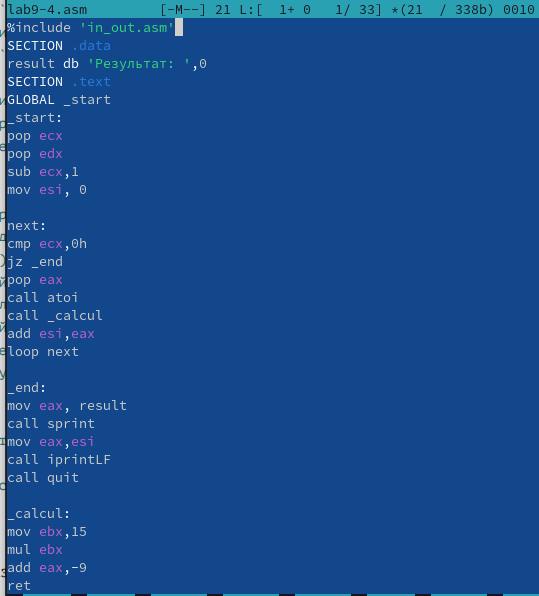
Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Теперь устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. [??]) 

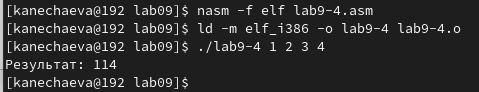
Адрес вершины стека хранится в регистре esp и по адресу вершины стека я смотрю количество аргументов командной строки (включая имя программы).(рис. [??]) Рис. 23. Информация о колличестве аргументов

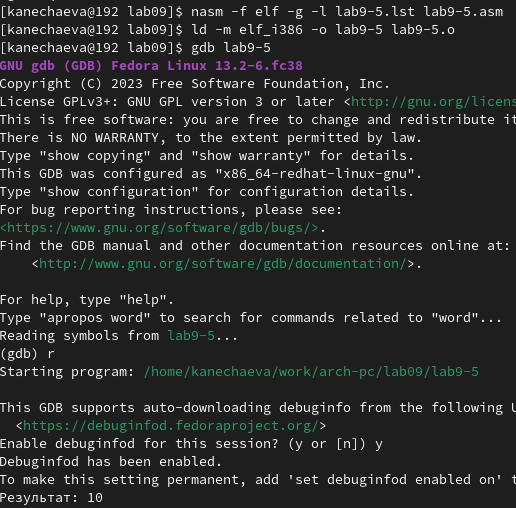
Как видно, число аргументов равно 5 – это имя программы lab09-3 и аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Теперь я смотрю остальные позиции стека: [esp+4] – адрес с именем программы, [esp+8] - адрес первого аргумента, [esp+12] – второго и т.д. (рис. [??]) 

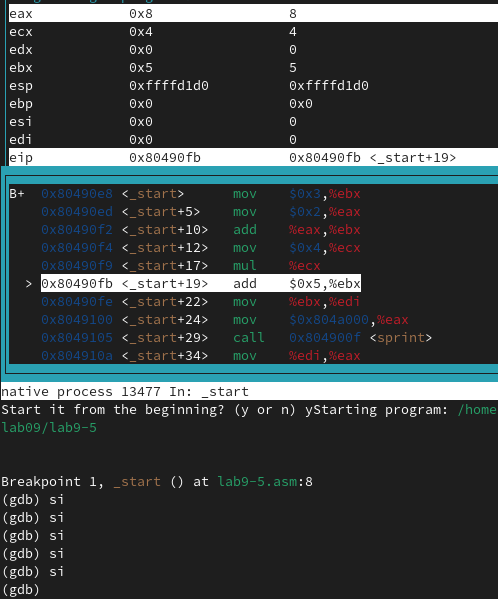
Шаг изменения адреса равен 4, т.к. размер ячейки памяти со значением, на которую указывает esp увеличивается на 4.

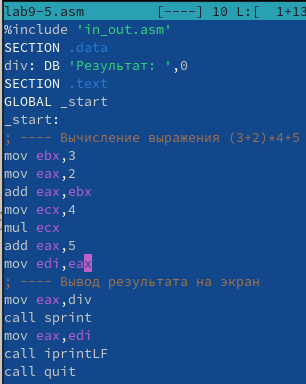
## 3.3 *3. Задание для самостоятельной работы*

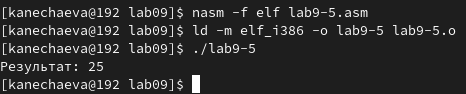
1. Требуется преобразовать программу из задания для самостоятельной работы лабораторной работы №8, реализовав вычисление значения функции 𝑓(𝑥) как подпрограмму. Для начала копирую файл lab8-5.asm в lab09 с именем lab9-4.asm и уже там начинаю работу над кодом. (рис. [??]) 

Создаю исполнительный файл и запускаю его. [??]) 

1. В листинге 9.3 приведена программа вычисления выражения (3 + 2) ∗ 4 + 5. При запуске данная программа дает неверный результат. [??]) 

Включаю режим псевдографики для более удобного анализа программы. [??]) 

Сначала в еbх записывается 3, в еах записывается 2, потом они складываются и в переменной ebx, в есх записывается 4. После этого идет mul ecx и регистр eax становится 8. Это произошло, т.к. по умолчанию идет умножение на то что записано в eax, а там 2, а 2\*4=8. Следовательно, надо записать результат сложения в переменную еах. И во всех дальнейших арифметических действиях я также заменяю edx на еах. [??]) 

Исправляю программу и запускаю ее снова. [??]) 

# 4 *4 Вывод*

При выполнении данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.

# 5 *5 Источники*

1. ТУИС – Архитектура ЭВМ – [Электронный ресурс] - https://esystem.rudn.ru/mod/resource/view.php?id=1030557