Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Мосолов Александр Денисович

Содержание

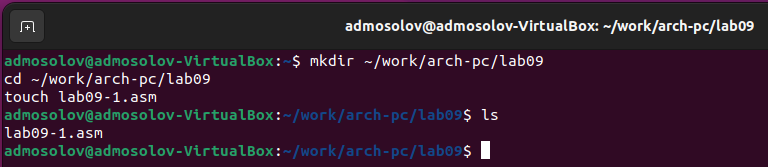
# 1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Выполнение лабораторной работы

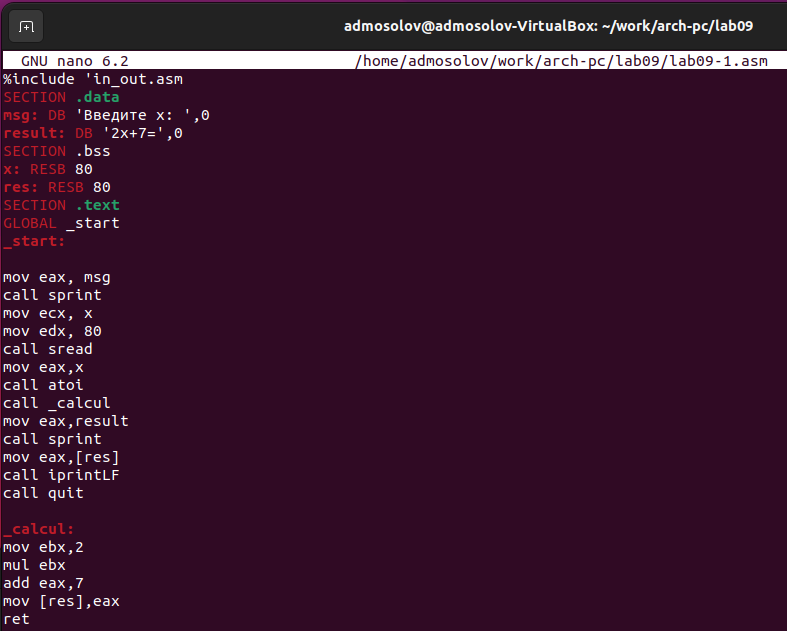
## 2.1 Реализация подпрограмм в NASM

С помощью *mkdir* создаем директорию, в которой будем работать во время выполнения лабораторной работы №9. Переходим в созданный каталог. С помощью touch создаем файл *lab09-1.asm.*



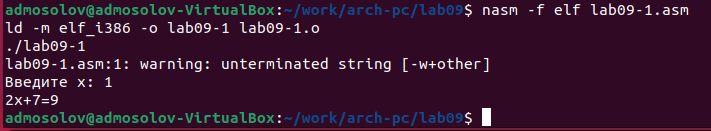
Создание каталога для работы

Внимательно изучив текст программы из листинга *9.1*, вводим его в файл *lab09-1.asm.*



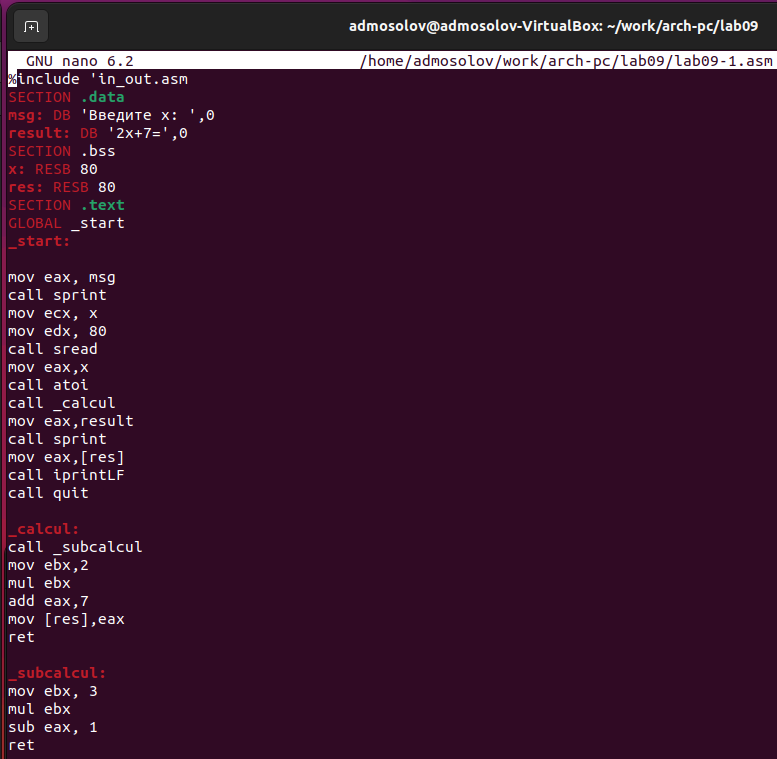
Текст программы lab09-1

Далее создаем исполняемый файл и запускаем его.



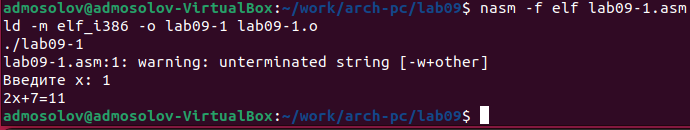
Создание исполняемого файла

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*.



Текст изменённой программы lab09-1

Далее создаем исполняемый файл и запускаем его.

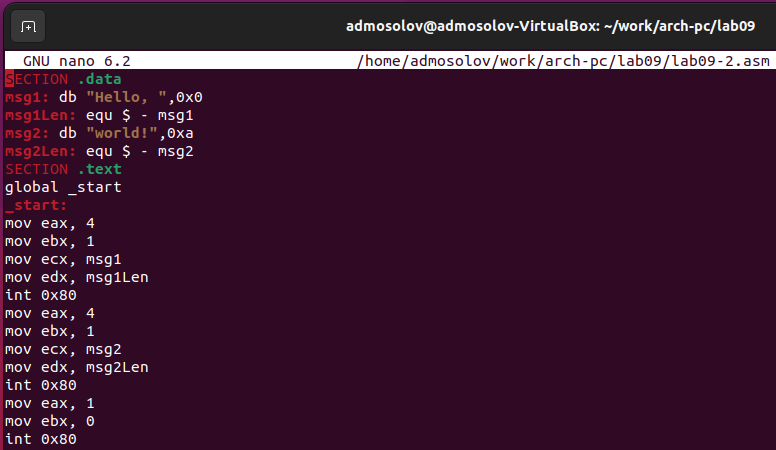


Создание испольняемого файла обновленной программы

## 2.2 Отладка программам с помощью GDB

Создаем файл *lab09-2.asm* в каталоге с помощью команды *touch lab09-2.asm*.

Открываем файл и заполняем его в соответствии с листингом *9.2*.



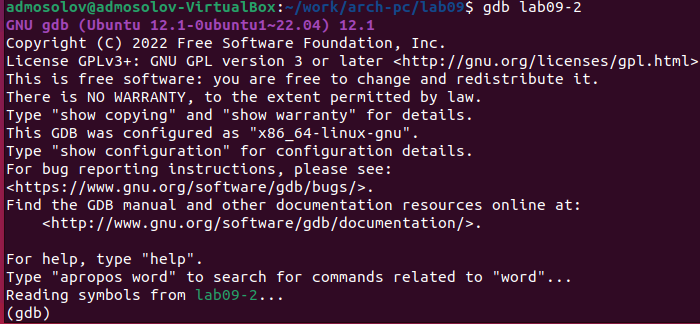
Текст программы lab09-2

Получим исполняемый файл. Для работы с *GDB* в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом *‘-g’*.

Получение исполняемого файла lab09-2

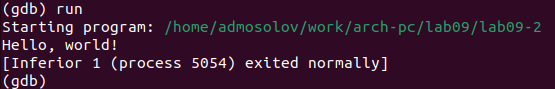
Получение исполняемого файла lab09-2

Загрузим исполняемый файл в отладчик *GDB*.



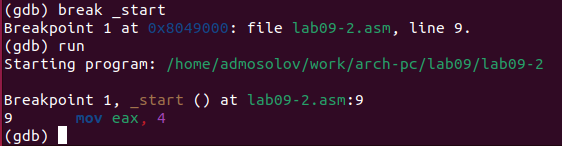
Загрузка исполняемого файла в отладчик GDB

Проверим работу программы, запустив ее в оболочке *GDB* с помощью команды *run*.



Проверка работы программы в оболочке GDB

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



Запуск программы с брэйкпоинтом на метке \_start

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



Дисассимилированный код программы

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

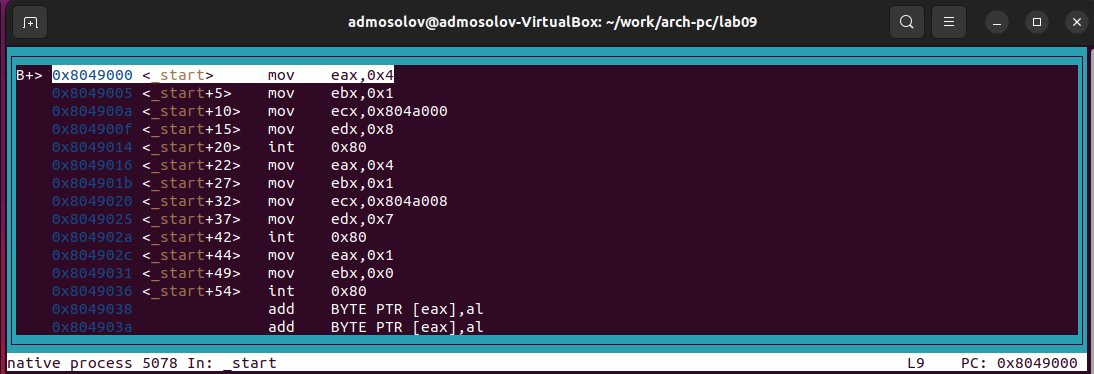


Intel’овский синтаксис

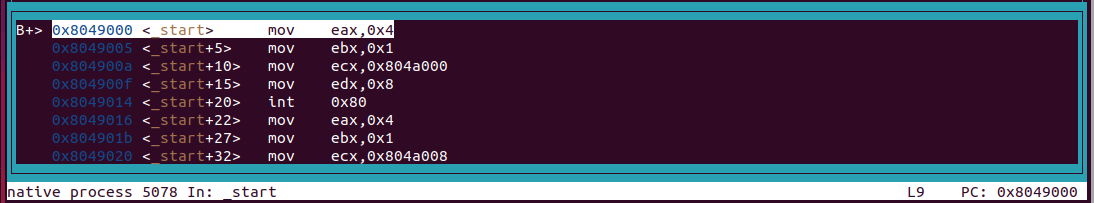
Различия отображения синтаксиса машинных команд в режимах ATT и Intel:

1. Разделители операндов также различаются: в ATT синтаксисе используются запятые, в то время как в Intel синтаксисе могут применяться запятые или косые черты (/).
2. В ATT синтаксисе порядок операндов отличается от Intel: исходный операнд указывается первым, а результирующий - вторым, в то время как в Intel синтаксисе порядок обычно прямой, с результирующим операндом первым и исходным операндом вторым.
3. Адреса в ATT синтаксисе указываются в круглых скобках, в то время как в Intel синтаксисе адреса указываются без скобок.
4. Префиксы размера операндов: В ATT синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (byte), “w” (word), “l” (long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”.
5. Знак операндов: В ATT синтаксисе операнды с позитивными значениями предваряются символом “$". В Intel синтаксисе операнды с позитивными значениями могут быть указаны без символа "$”.

Включим режим псевдографики для более удобного анализа программы.



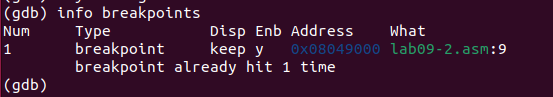
Режим псевдографики №1



Режим псевдографики №2

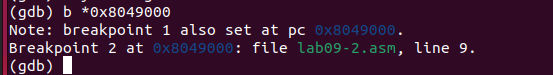
### 2.2.1 Добавление точек останова

Проверим установленные точки останова с помощью команды *info breakpoints*.



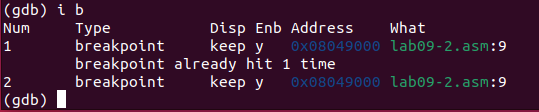
Информация о точках останова

Определим адрес предпоследней инструкции *(mov ebx,0x0)* и установим точку останова.



Установка точки останова

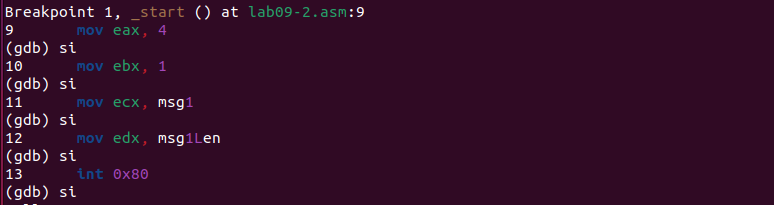
Посмотрим информацию о всех установленных точках останова.



Информация о точках останова

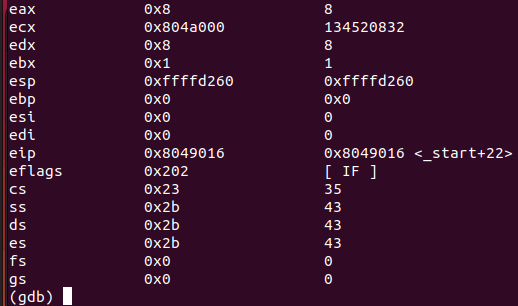
### 2.2.2 Работа с данными программы в GDB

Выполним 5 инструкций *stepi*.



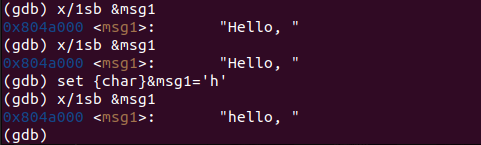
Выполнение 5 инструкций stepi

Посмотрев содержимое регистров с помощью команды *info registers* (или *i r*), понимаем, что во время выполнения команд менялись регистры: *ebx, ecx, edx, eax, eip*.



Информация о регистрах

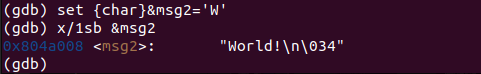
С помощью команды x & также можно посмотреть содержимое переменной. Просмотрим значение переменной *msg1* по имени и изменим первый символ слова на *h*.



Изменяем первого символа переменной msg1

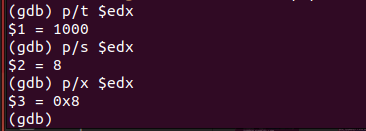
Можно смотреть значение пременой не только по имени, но и по адресу в памяти.

Поменяем первый символ слова переменной *msg2* на *W*.



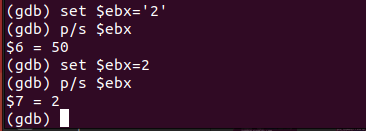
Изменяем первого символа переменной msg2

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



Выводим значение регистра edx

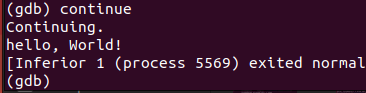
Изменим регистр ebx.



Изменение регистра ebx

Выводятся разные значения, команда без кавычек присваивает регистру вводимое значение.

Завершим выполнение программы с помощью команды continue (сокращенно c).



Завершение программы

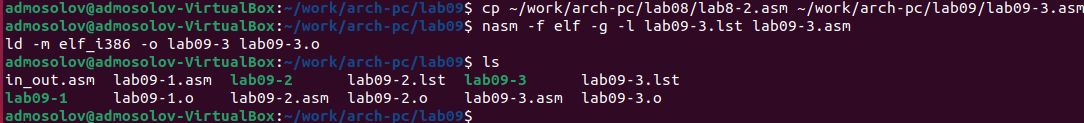
Выйдем из *GDB* с помощью команды *quit* (сокращенно *q*).

Выход из GDB

Выход из GDB

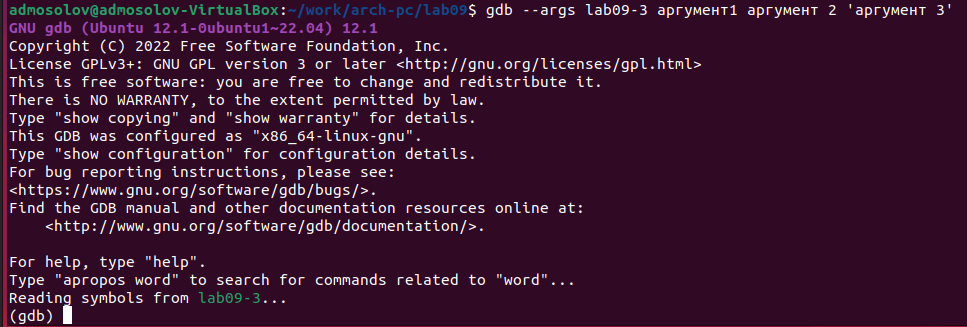
### 2.2.3 Обработка аргументов командной строки в GDB

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*. И создадим исполняемый файл.



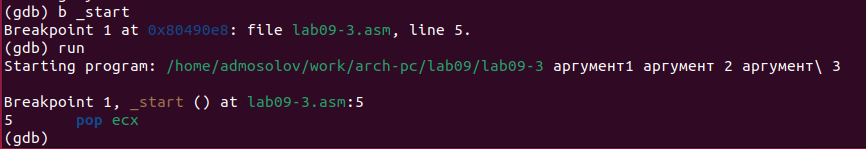
Копирование и создание исполняемого файла

Загрузим исполняемый файл в отладчик, указав ключ *–args* и аргументы.



Загрузка исполняемого файла в отладчик

Для начала установим точку останова перед первой инструкцией в программе и запустим ее.



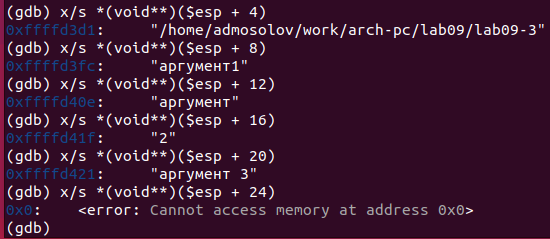
Установка точки останова и запуск программы

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы).

Адрес вершины стека

Адрес вершины стека

Просмотрим остальные позиции стека.



Просмотр остальных позиций стека

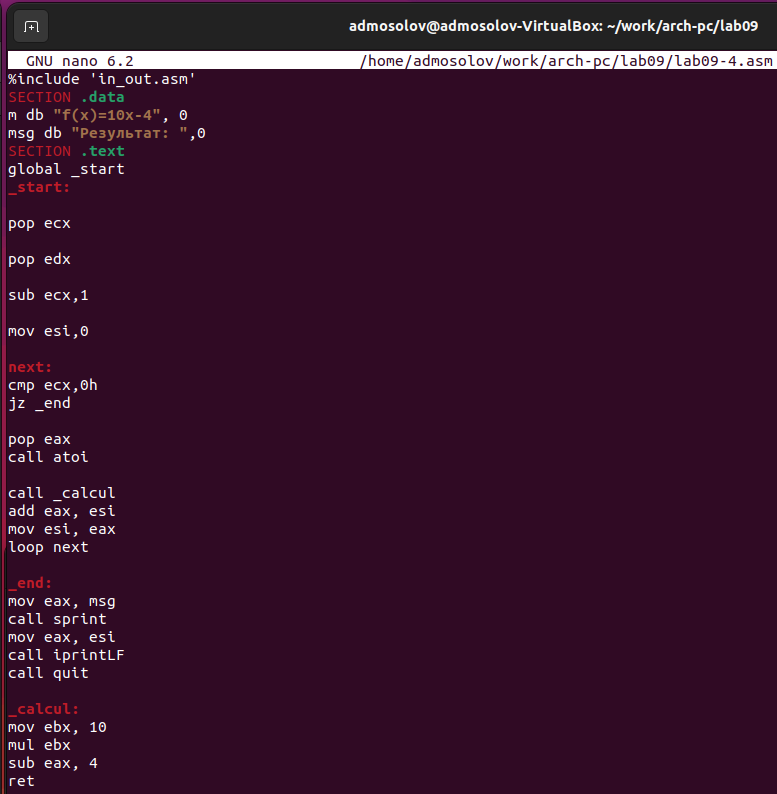
Шаг изменения адреса равен 4, т.к. в большинстве архитектур процессоров размер слова (или размер указателя) составляет 4 байта. Это означает, что каждый раз, когда мы обращаемся к следующему элементу в стеке, мы увеличиваем адрес на 4. Шаг изменения адреса равен 4 для обеспечения корректного доступа к данным в стеке.

## 2.3 Выполнение заданий для самостоятельной работы

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

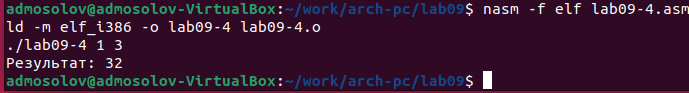
Создаем файл для выполнения задания с помощью команды *touch lab09-4.asm*.

Вставим и отредактируем текст программы из лабораторной работы №8 с добавлением подпрограммы.



Текст отредактированной программы

Далее создаем исполняемый файл и запускаем его.

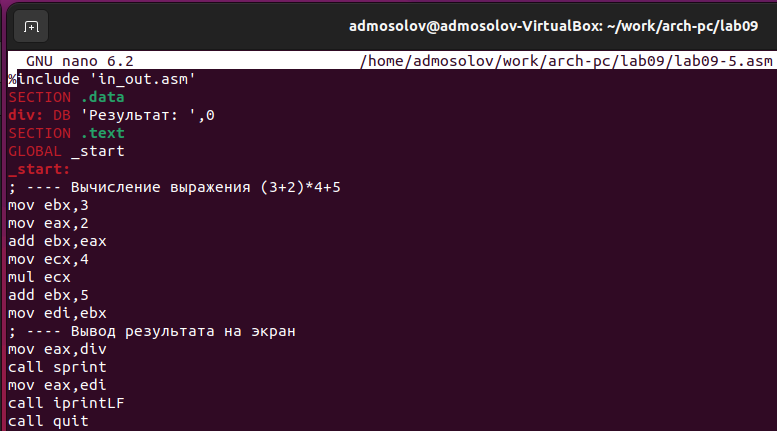


Создание и запуск исполняемого файла lab09-4

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.

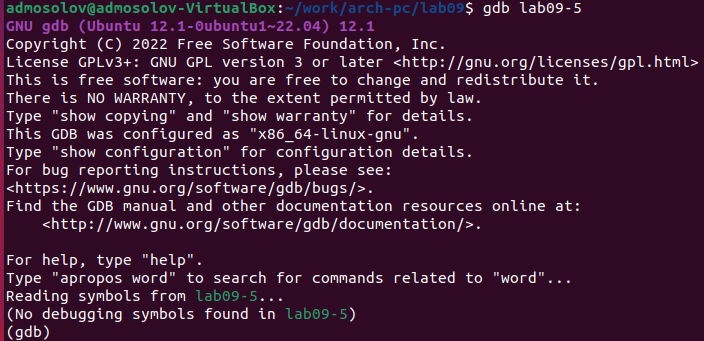
Создаем файл для выполнения задания с помощью команды *touch lab09-5.asm*.

Вставим программу из листинга *9.3*.



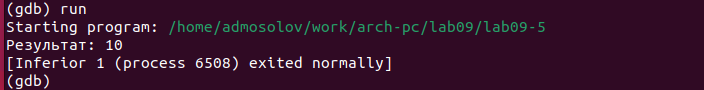
Текст программы листинга 9.3

Создаем исполняемый файл и запускаем его в отладчике *GDB*. Смотрим на изменение регистров по ходу программы.



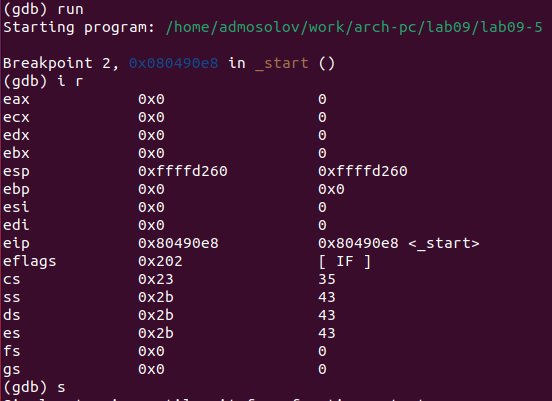
Загрузка файла в отладчик

Запустим программу с помощью *run*.



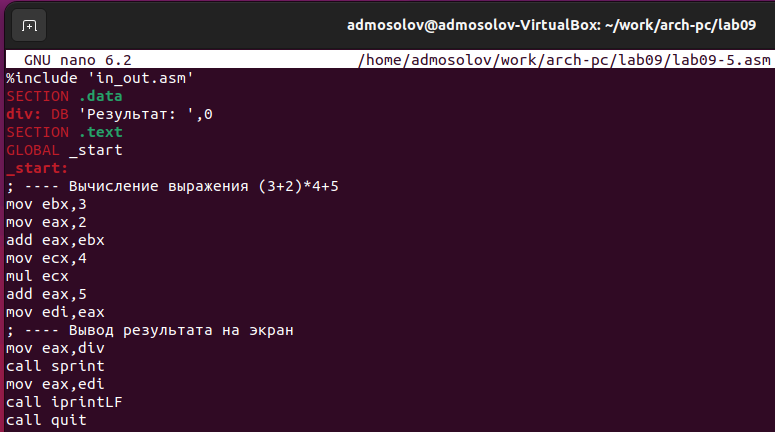
Запуск программы

Смотрим изменение регистров.



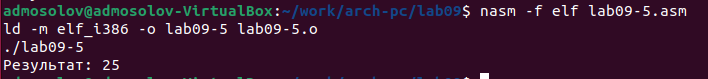
Изменение регистров

Корректируем программу, обнаружив ошибку неправильной записи регистров.



Текст изменённой программы lab09-5

Далее создаем исполняемый файл и запускаем его, проверяем правильность выполненной программы.



Создание и запуск исполняемого файла

# 3 Выводы

В ходе выполнения лабораторной работы мы приобрели навыки написания программ с использованием подпрограмм, познакомились с методами отладки при помощи *GDB*.