Отчёт по лабораторной работе №9

дисциплина: Архитектура компьютера

Люкшина Влада Алексеевна

Содержание

# 1)Цель работы

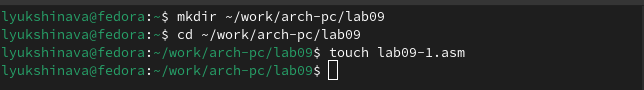
Приобреcти навыки написания программ с использованием подпрограмм. Познакомиться с методами отладки при помощи GDB и его основными возможностями.

# 2)Задание

Написать программы с использованием вызова подпрограммы, проанализировать их. Преобразовать программу из лабораторной работы №8.

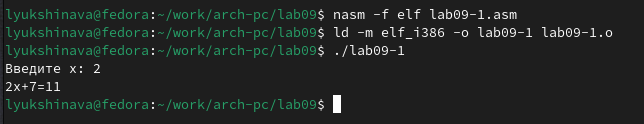
# 3)Выполнение лабораторной работы

## 3.1) Создаем каталог для выполнения лабораторной работы № 9, переходим в него и создаем файл lab09-1.asm.



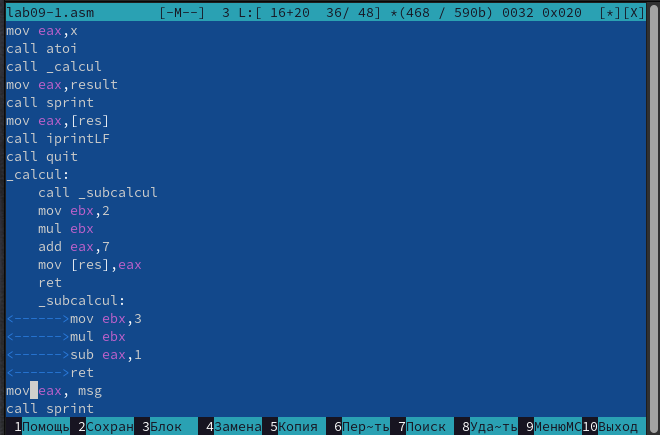
Создаем каталог, переходим в него и создаем файл

## 3.2) Вводим в файл lab09-1.asm текст программы из листинга 9.1. Создаем исполняемый файл и проверяем его работу.



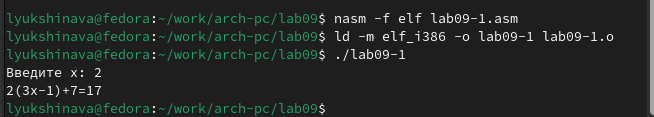
Вводим текст, создаем файл и проверяем его работу

## 3.3) Изменяем текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul.



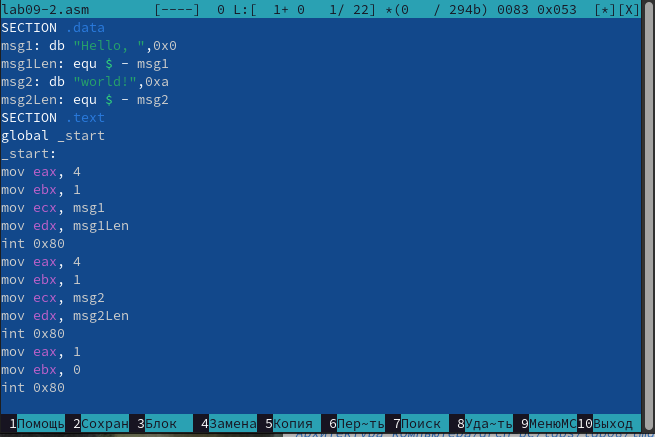
Изменяем текст программы

## 3.4) Проверяем работу файла.



Проверяем

## 3.5) Создаем файл lab09-2.asm с текстом программы из Листинга 9.2.



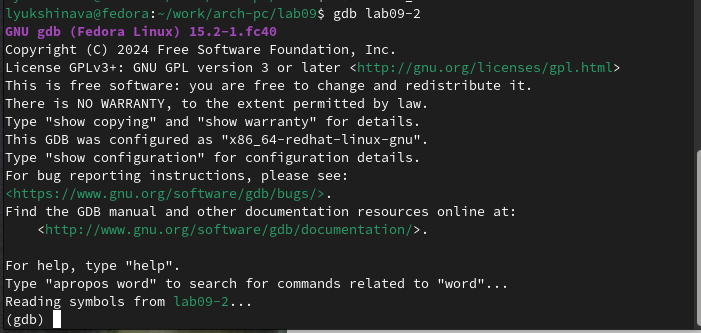
Создаем файл и вводим текст из листинга

## 3.6) Получаем исполняемый файл. Для работы с GDB в исполняемый файл добавляем отладочную информацию, используя ключ -g.

Получаем исполняемый файл

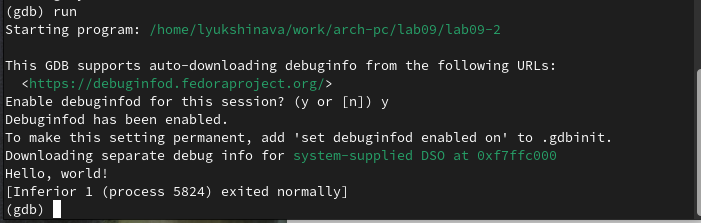
Получаем исполняемый файл

## 3.7) Загружаем исполняемый файл в отладчик gdb.



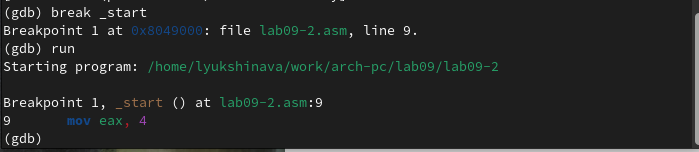
Загружаем файл

## 3.8) Проверяем работу программы, запустив ее в оболочке GDB с помощью команды run.



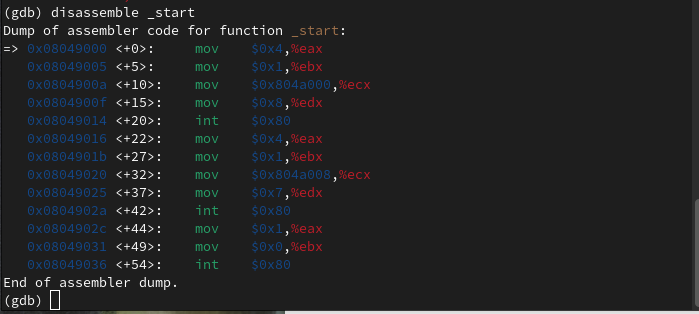
Проверяем

## 3.9) Устанвливаем брейкпоинт на метку \_start и запускаем программу.



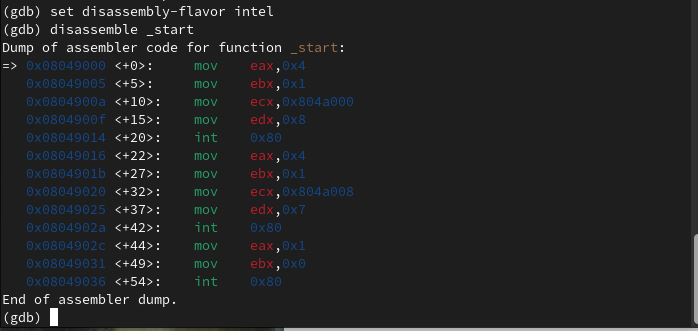
Устанавливаем брейкпоинт и запускаем программу

## 3.10) Просматриваем дисассимилированный код программы с помощью команды disassemble начиная с метки \_start.



Просматриваем код

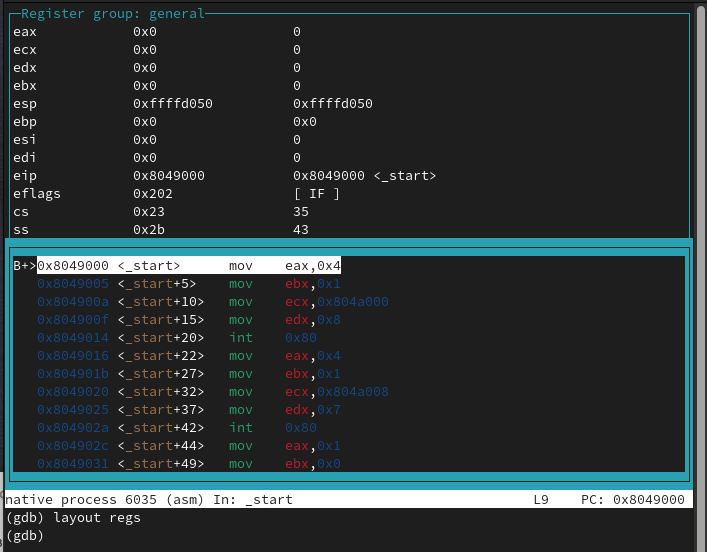
## 3.11) Переключаемся на отображение команд с Intel’овским синтаксисом с помощью команды set disassembly-flavor intel.



Переключаемся и смотрим программу

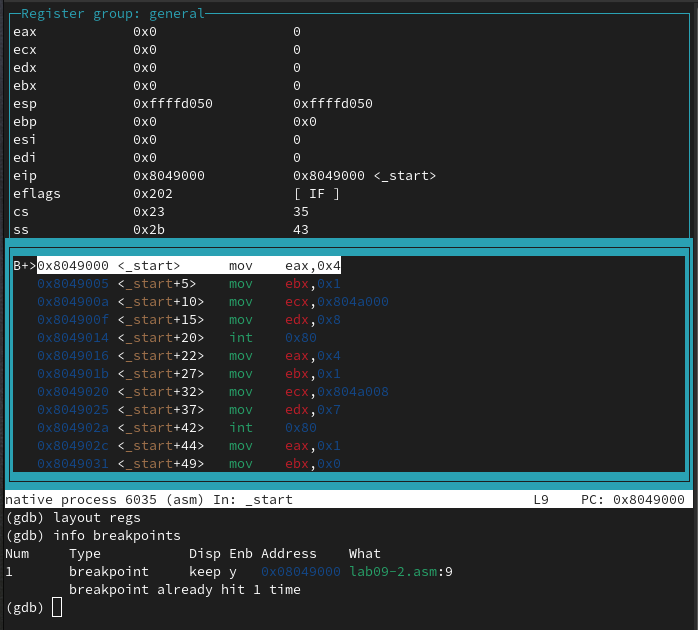
Различия отображения синтаксиса машинных команд в режимах ATT и Intel: 1) В ATT синтаксисе обозначение регистра начинается с символа “%”. В Intel синтаксисе обозначение регистра может начинаться с символа “R” или “E”. 2) В ATT синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный - вторым. 3) В ATT синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок. 4) В ATT синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (byte). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”. 5) В ATT синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

## 3.12) Включаем режим псевдографики для более удобного анализа программы.



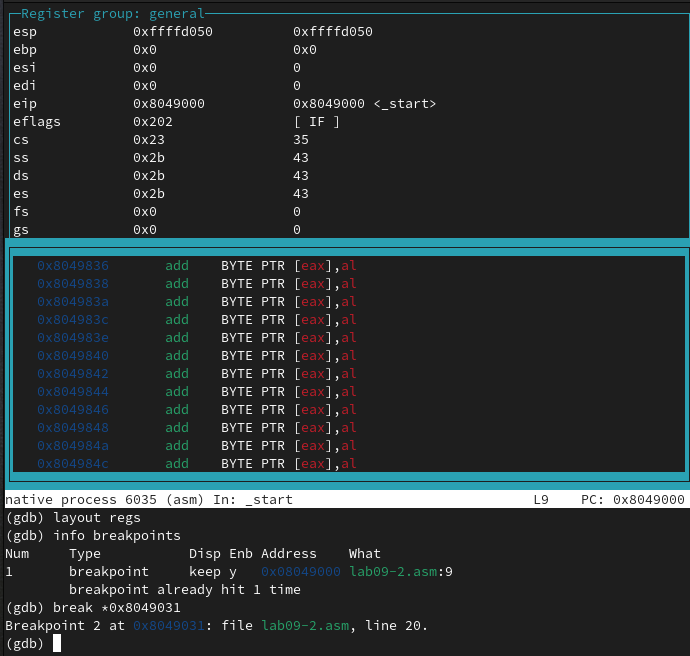
Включаем режим псевдографики

## 3.13) Проверяем, установлена ли точка останова по имени метки(\_start).



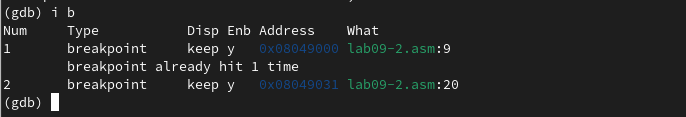
Проверяем

## 3.14) Определяем адрес предпоследней инструкции (mov ebx,0x0) и устанавливаем точку останова.



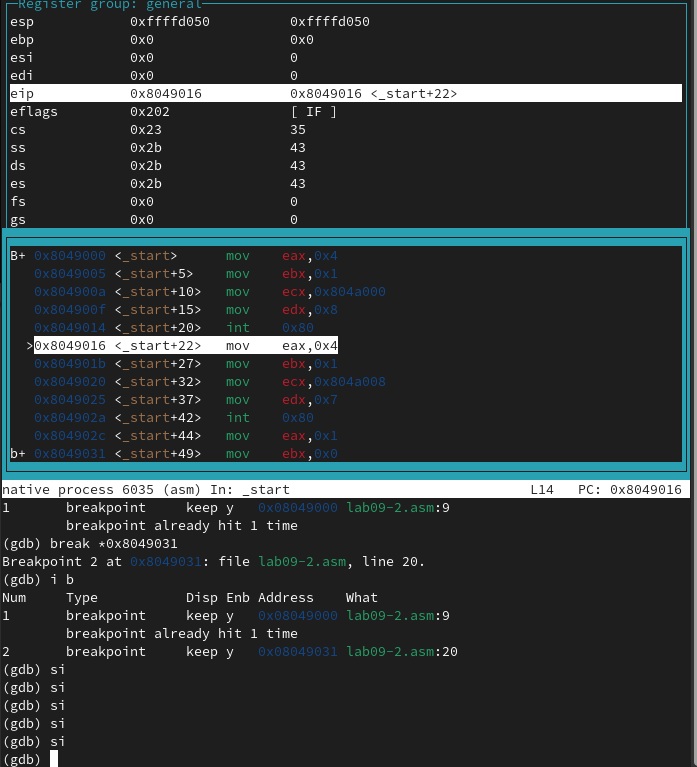
Устанавливаем точку останова

## 3.15) Cмотрим информацию о всех установленных точках останова.



Смотрим

## 3.16) Выполняем 5 инструкций с помощью команды stepi (или si).

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

## 3.17) Cмотрим значение переменной msg1 по имени.

Смотрим значение переменной

Смотрим значение переменной

## 3.18) Смотрим значение переменной msg2 по адресу.

Смотрим значение переменной

Смотрим значение переменной

## 3.19) Изменяем первый символ переменной msg1.



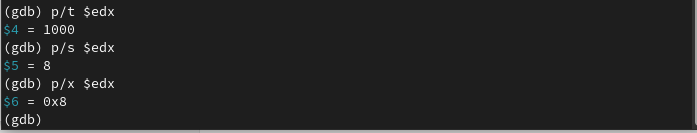
Изменяем символ

## 3.20) Изменяем символ во второй переменной msg2.



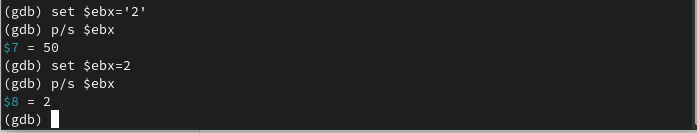
Изменяем символ

## 3.21) Выводим в различных форматах значение регистра edx.

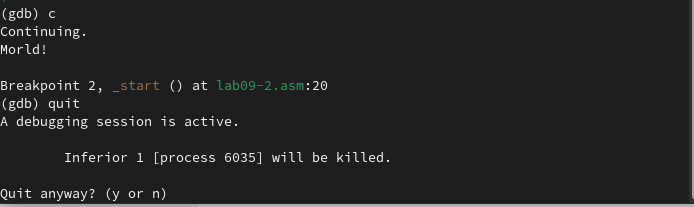


Выводим значение

## 3.22) С помощью команды set изменяем значение регистра ebx.

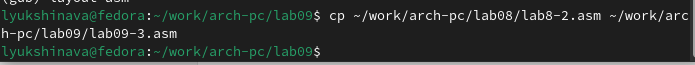
 Команда без кавычек присваивает регистру вводимое значение, поэтому при выведении значения регистра мы получаем разные результаты.

## 3.23) Завершаем выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и выходим из GDB с помощью команды quit (сокращенно q).



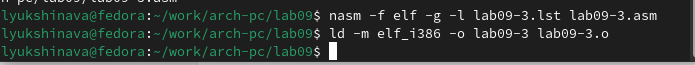
Завершаем выполнение программы

## 3.24) Копируем файл lab8-2.asm в файл с именем lab09-3.asm.



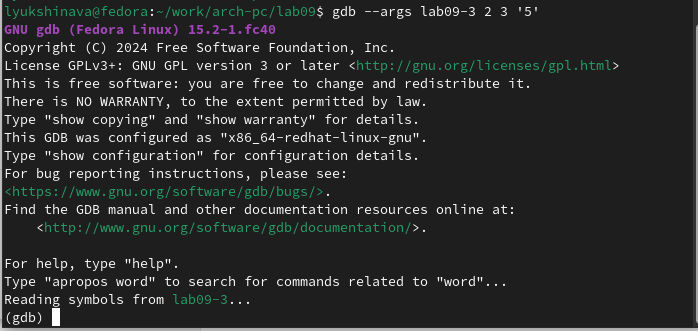
Копируем файл

## 3.25) Создаем исполняемый файл.



Создаем исполняемый файл

## 3.26) Загружаем исполняемый файл в отладчик, указав аргументы.



Создаем исполняемый файл

## 3.27) Устанавливаем точку останова перед первой инструкцией в программе и запускаем ее.



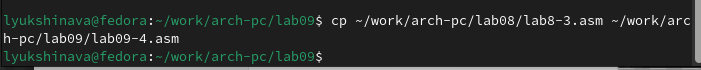
Устанавливаем точку и запускаем

## 3.28) Просматриваем позиции стека по разным адресам.

 Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 4 байта.

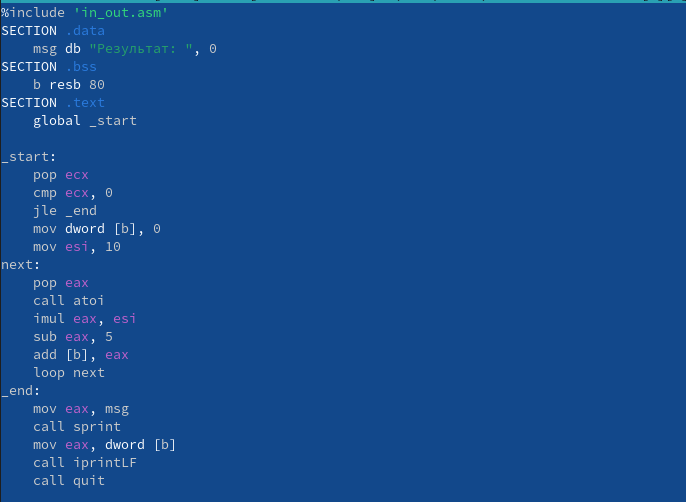
# 4) Самостоятельная работа

## 4.1) Копируем файл lab8-3.asm в файл lab09-4.asm



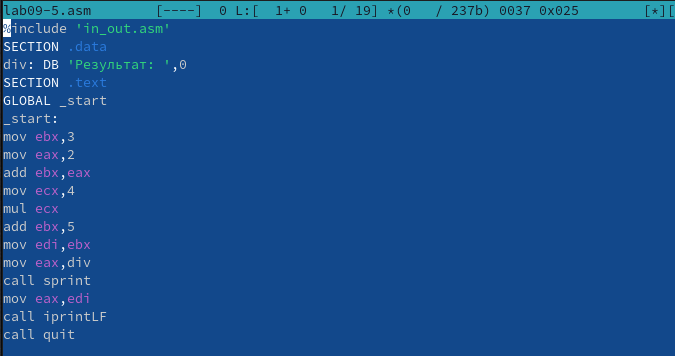
Копируем

## 4.2) Изменяем файл, реализуя вычисления значения через подпрограмму(вариант 3, 10х-5)



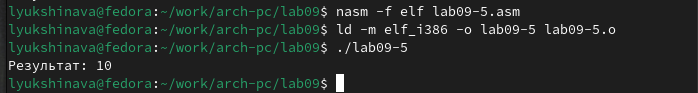
Изменяем файл

## 4.3) Создаем файл lab09-5.asm и записываем туда текст из листинга.



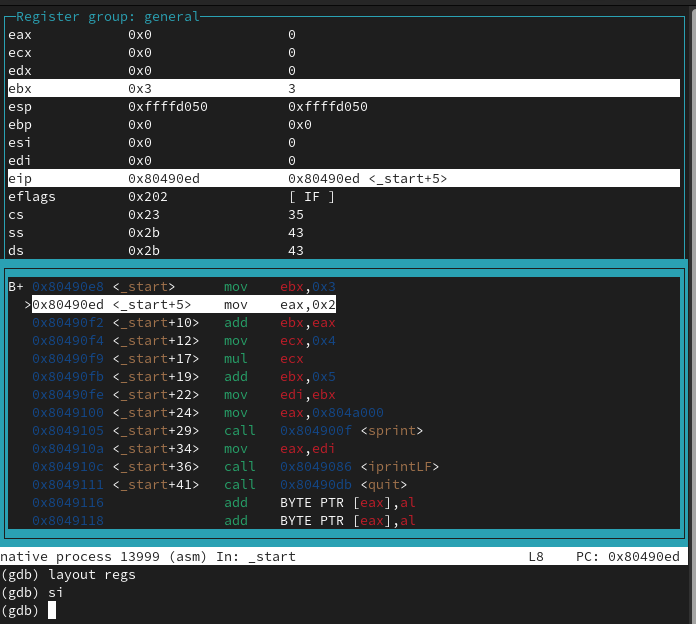
Создаем файл

## 4.4) Создаем исполняемый файл и проверяем его работу.



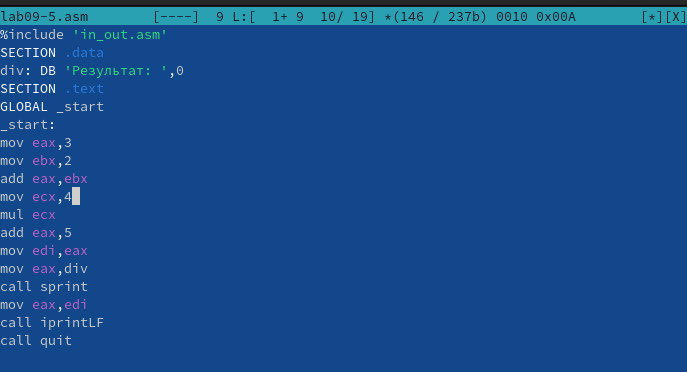
Проверяем. Программа работает неправильно

## 4.5) Создаем исполняемый файл, запускаем его в отладчике GDB и ищем ошибку.



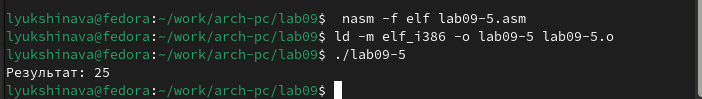
Проверяем регистры

## 4.6) Изменяем программу.



Изменяем программу

## 4.7) Создаем исполняемый файл и проверяем его работу.



Проверяем

# 5)Выводы

Мы познакомились с методами отладки при помощи GDB и его возможностями.