Отчёта по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Мантуров Татархан Бесланович

Содержание

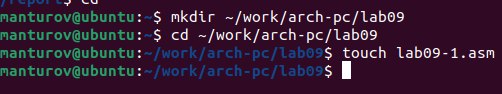
# 1 Цель работы

Познакомиться с методами отладки при помощи GDB, его возможностями.

# 2 Выполнение лабораторной работы

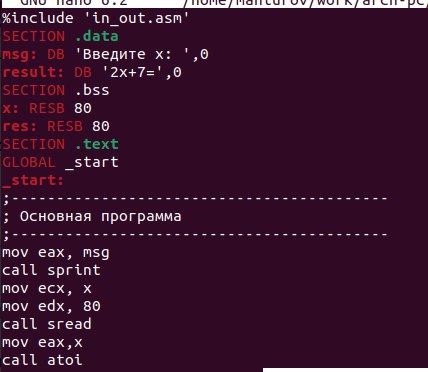
## 2.1 Реализация подпрограмм в NASM

Создаем каталог для программ ЛБ9, и в нем создаем файл (рис. ??).



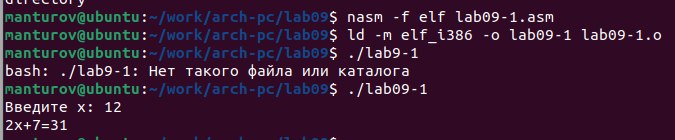
Создаем каталог с помощью команды mkdir и файл с помощью команды touch

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. ??).



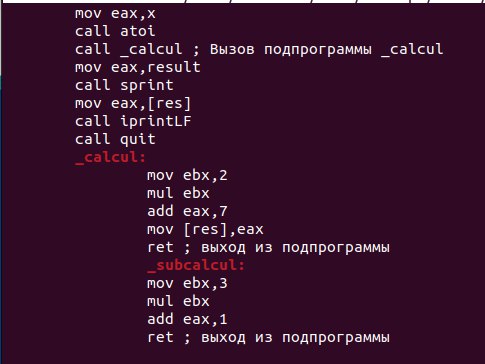
Заполняем файл

Создаем исполняемый файл и запускаем его (рис. ??).



Запускаем файл и проверяем его работу

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму(по условию) (рис. ??).



Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его (рис. ??).

Запускаем файл и смотрим на его работу

Запускаем файл и смотрим на его работу

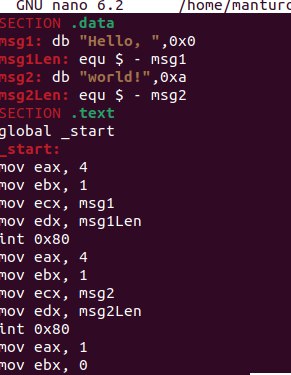
## 2.2 Отладка программам с помощью GDB

Создаем новый файл в каталоге(рис. ??).

Создаем файл

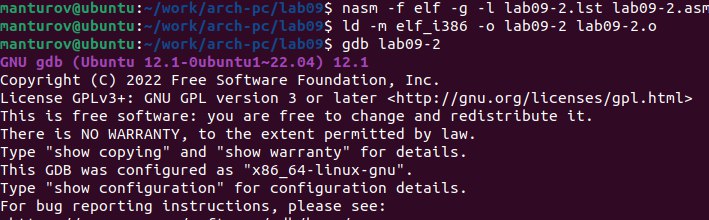
Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. ??).



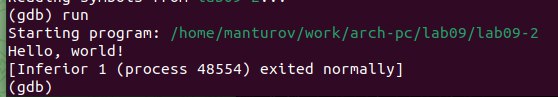
Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. ??).



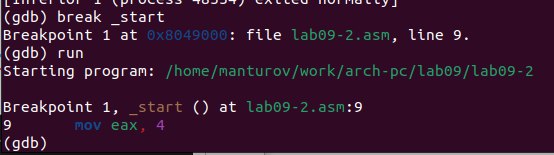
Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. ??).



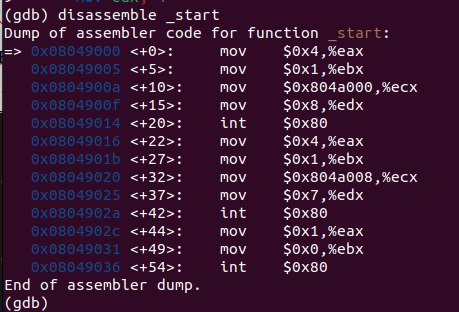
Запускаем программу командой run

Устанавливаем брейкпоинт на метку \_start и запускаем программу (рис. ??).



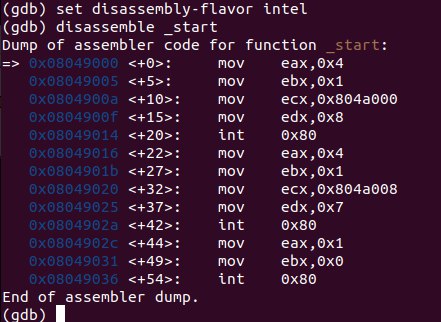
Запускаем программу с брейкпоином

Смотрим дисассимилированный код программы с помощью команды disassemble, начиная с метки \_start(рис. ??).



Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel’овским синтаксисом (рис. ??).



Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах ATT и Intel:

1.Порядок операндов: 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.

2.Разделители: В ATT синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

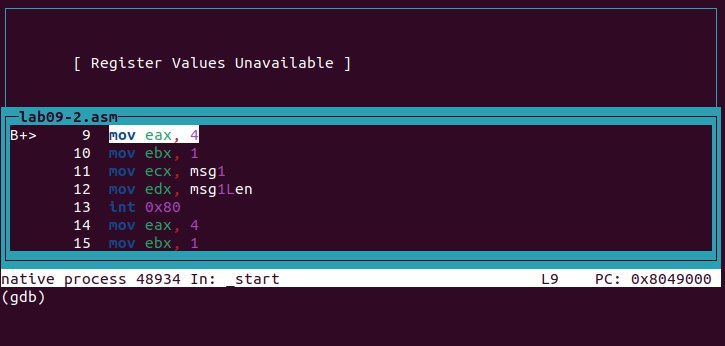
3.Префиксы размера операндов: В ATT синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (byte), “w” (word), “l” (long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”.

4.Знак операндов: В ATT синтаксисе операнды с позитивными значениями предваряются символом “$". В Intel синтаксисе операнды с позитивными значениями могут быть указаны без символа "$”.

5.Обозначение адресов: В ATT синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.

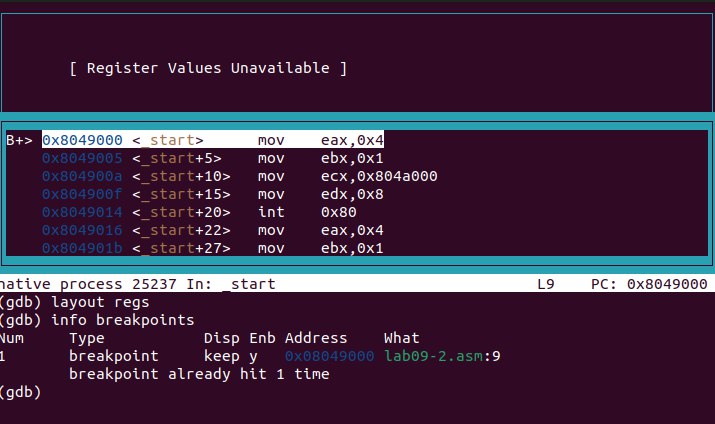
6.Обозначение регистров: В ATT синтаксисе обозначение регистра начинается с символа “%”. В Intel синтаксисе обозначение регистра может начинаться с символа “R” или “E” (например, “%eax” или “RAX”).

Включаем режим псевдографики (рис. ??).



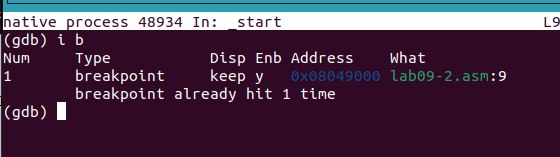
Включаем отображение регистров, их значений и результат дисассимилирования программы

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. ??).



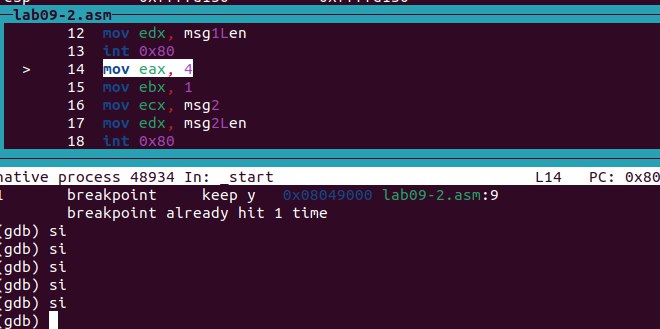
Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова (рис. ??).



Смотрим информацию

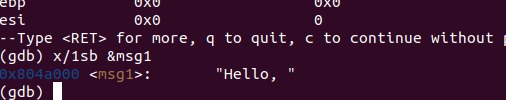
Выполняем 5 инструкций командой si (рис. ??).



Отслеживаем регистры

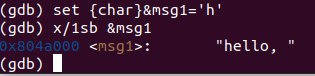
Во время выполнения команд менялись регистры: ebx, ecx, edx,eax, eip.

Смотрим значение переменной msg1 по имени (рис. ??).



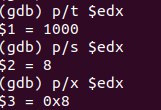
Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. ??).



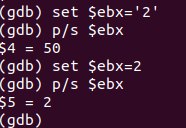
Меняем символ

Смотрим значение регистра edx в разных форматах (рис. ??).



Смотрим значение регистра

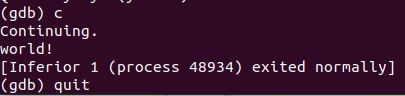
Изменяем регистор ebx (рис. ??).



Изменяем регистор командой set

Выводится разные значения, так как команда без кеавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB (рис. ??).



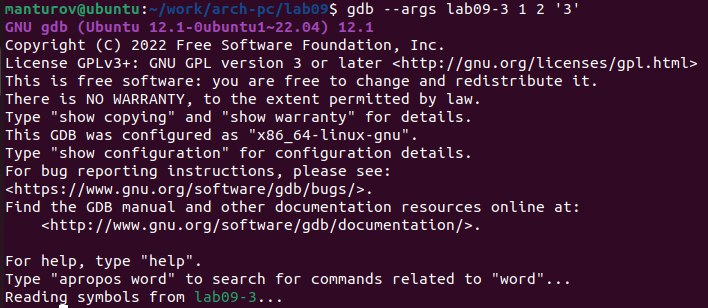
Прописываем команды c и quit

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. ??).

Копируем файл

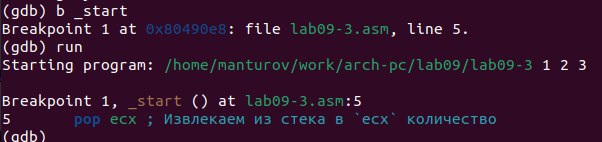
Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. ??).



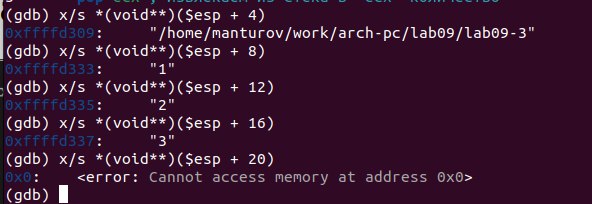
Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. ??).



Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. ??).



Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

##Задание для самостоятельной работы

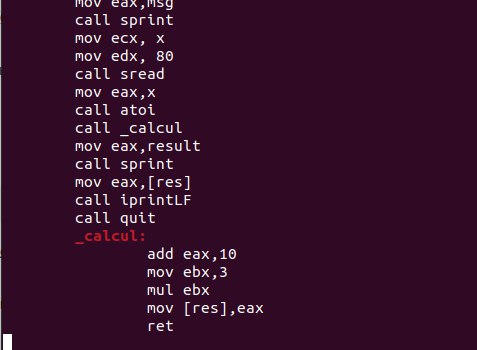
###Задание 1

Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем samos9.asm (рис. ??).

Копируем файл

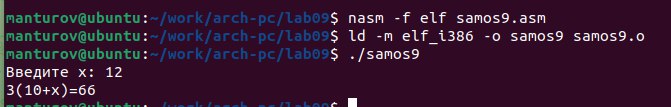
Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. ??).



Изменяем файл

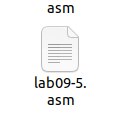
Создаем исполняемый файл и запускаем его (рис. ??).



Проверяем работу программы

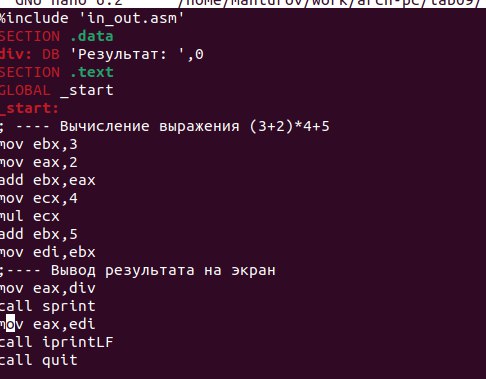
###Задание 2

Создаем новый файл в дирректории (рис. ??).



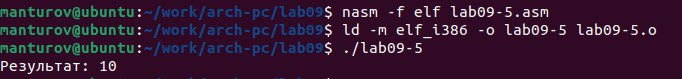
Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. ??).



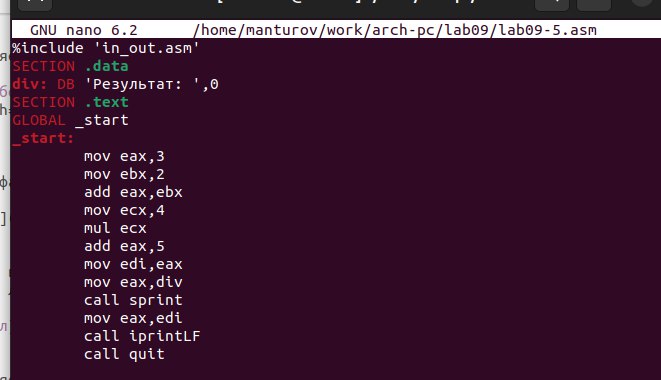
Изменяем файл

Создаем исполняемый файл и запускаем его (рис. ??).



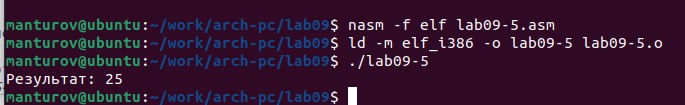
Создаем и смотрим на работу программы(работает неправильно)

Изменяем программу для корректной работы (рис. ??).



Меняем файл

Создаем исполняемый файл и запускаем его (рис. ??).



Создаем и запускаем файл(работает корректно)

# 3 Выводы

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