Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

НовичковМаксим Алексеевич

Содержание

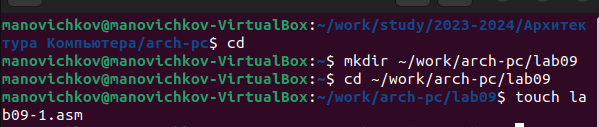
# 1 Цель работы

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

# 2 Выполнение лабораторной работы

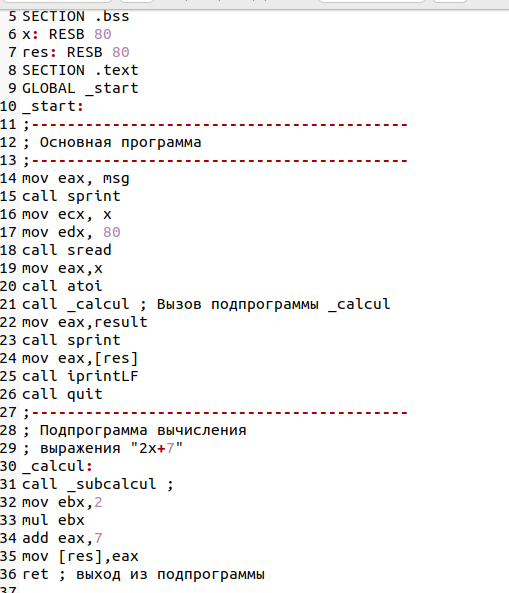
## 2.1 Реализация подпрограмм в NASM

1. С помощью утилиты mkdir создаем директорию, в которой будем создавать файлы с программами для лабораторной работы №9. Переходим в созданный каталог с помощью утилиты cd. С помощью утилиты touch создаем файл lab09-1.asm (рис. [??]).



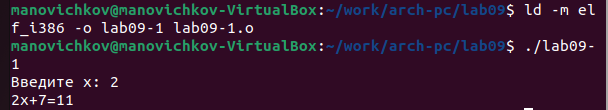
Создание необходимой директории и файла

1. Внимательно изучив текст программы из листинга 9.1, вводим его в файл lab09-1.asm (рис. [??]).



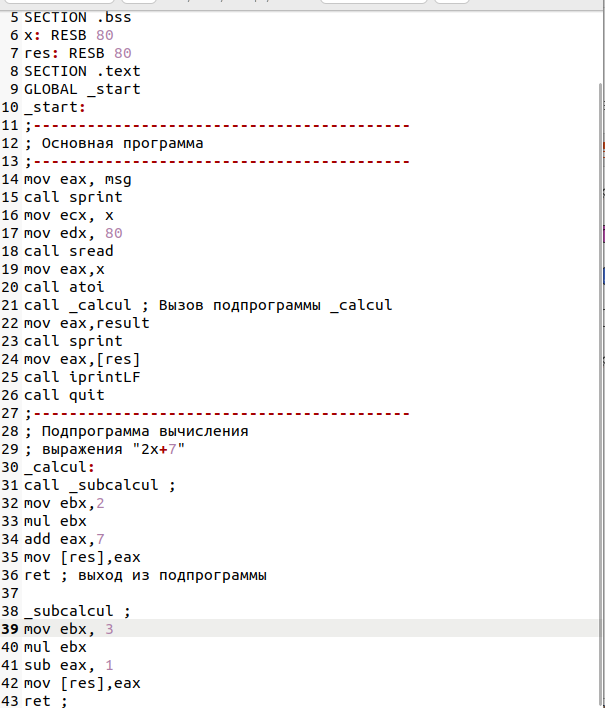
Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [??], .



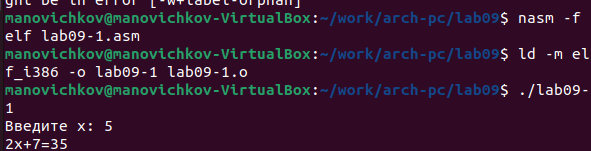
Запуск исполняемого файла

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 (рис. [??]).



Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [??]).



Запуск исполняемого файла

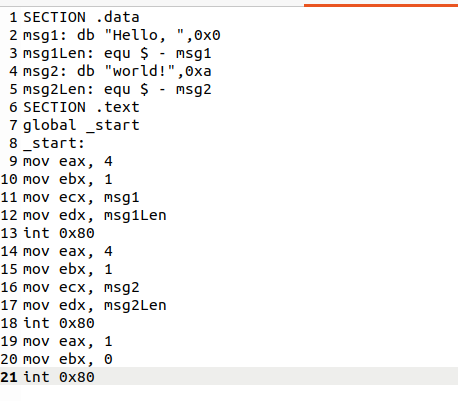
## 2.2 Отладка программам с помощью GDB

Создаем новый файл lab09-2.asm в каталоге (рис. [??]).

Создание необходимого файла

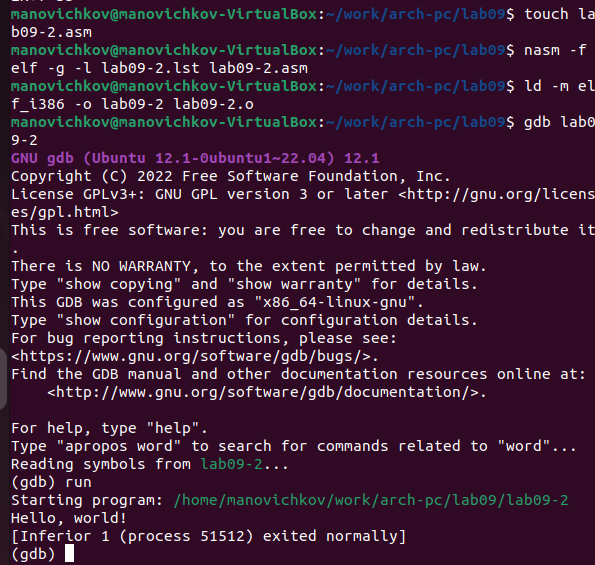
Создание необходимого файла

Открываем файл и заполняем его в соответствии с листингом 9.2 (рис. [??]).



Редактирование файла

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



Проверка работы программы

Для более подробного анализа программы установим брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустим её (рис. [??]).

Запуск программы с брэйкпоинтом

Запуск программы с брэйкпоинтом

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. [??]).

![Дисассимилированный код программы](image/Снимок экрана от 2023-12-09 21-57-46.png{#fig:010 width=70%}

Переключимся на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. [??]).

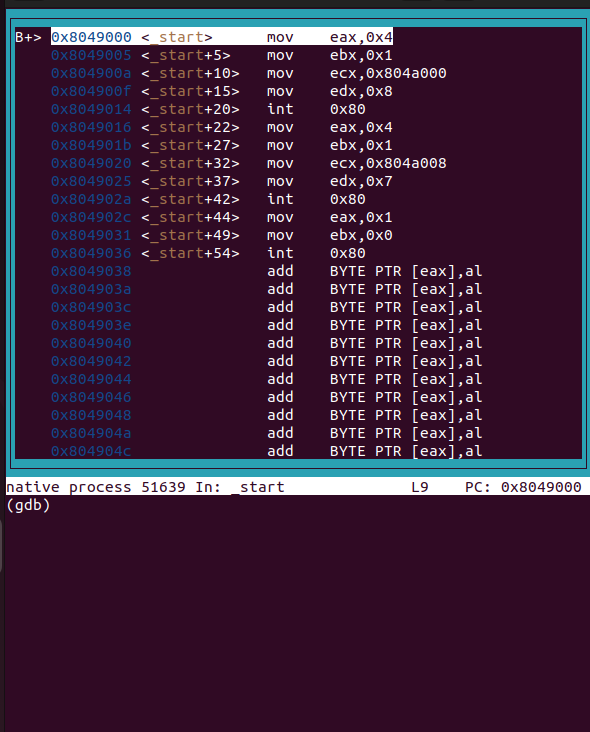


Синтаксис Intel

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”).

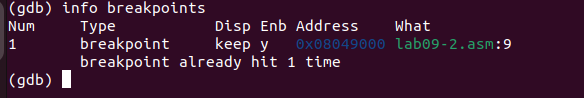
Включим режим псевдографики для более удобного анализа программы (рис. [??]).



Режим псевдографики

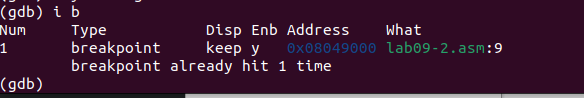
### 2.2.1 Добавление точек останова

Проверим установленные точки останова с помощью команды info breakpoints (рис. [??]).



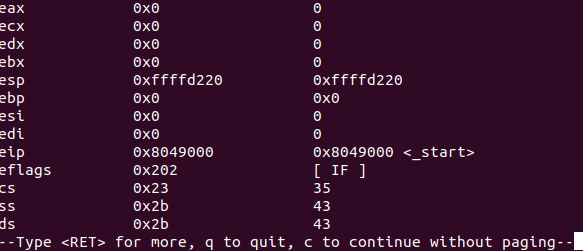
Информация о точках останова

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



Установка точки останова

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



Информация о точках останова

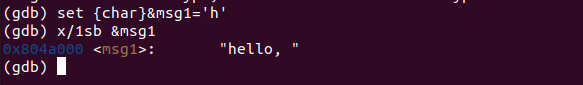
### 2.2.2 Работа с данными программы в GDB

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

Информация о регистрах

Информация о регистрах

С помощью команды x & также можно посмотреть содержимое переменной. Посмотрим значение переменной msg1 по имени (рис. [??]).



Значение переменной

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



Изменение первого символа переменной msg1

Изменим первый символ переменной msg2 (рис. [??]).

Изменение первого символа переменной msg2

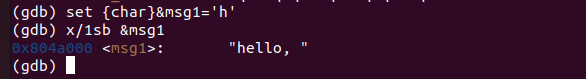
Изменение первого символа переменной msg2

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

Значение регистра edx

Значение регистра edx

Изменим регистр ebx (рис. [??]).



Изменение регистра ebx

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

Завершим выполнение программы с помощью команды continue (сокращенно c) и выйдем из GDB с помощью команды quit (сокращенно q) (рис. [??]).

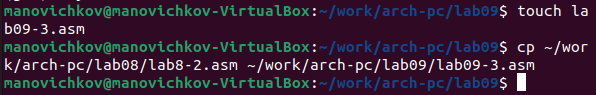
|  |
| --- |
| Завершение программы и выход из GDB |

Завершение программы и выход из GDB

## 2.3 Выполнение заданий для самостоятельной работы

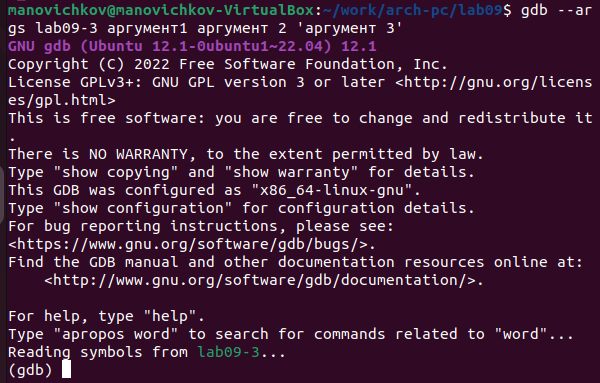
1. Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму.

Вставляем отредактированную программу из лабораторной работы №8 с добавлением подпрограммы (рис. [??]).



Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [??], [??]).



Создание исполняемого файла

# 3 Выводы

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