Лабораторная работа №10

Архитектура компьютера

Косолапов Матвей Эдуадович

Содержание

# 1 Цель работы

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

# 2 Задание

# 3 Теоретическое введение

# 4 Выполнение лабораторной работы

1. Cоздаём каталог lab10, в нём создаём файл lab10-1.asm(рис. 1):

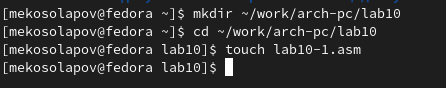


Рис. 1: Создание каталога lab10 и файла lab10-1.asm

1. Переносим в файл программу вычисления функции из листинга №1(рис. 2):

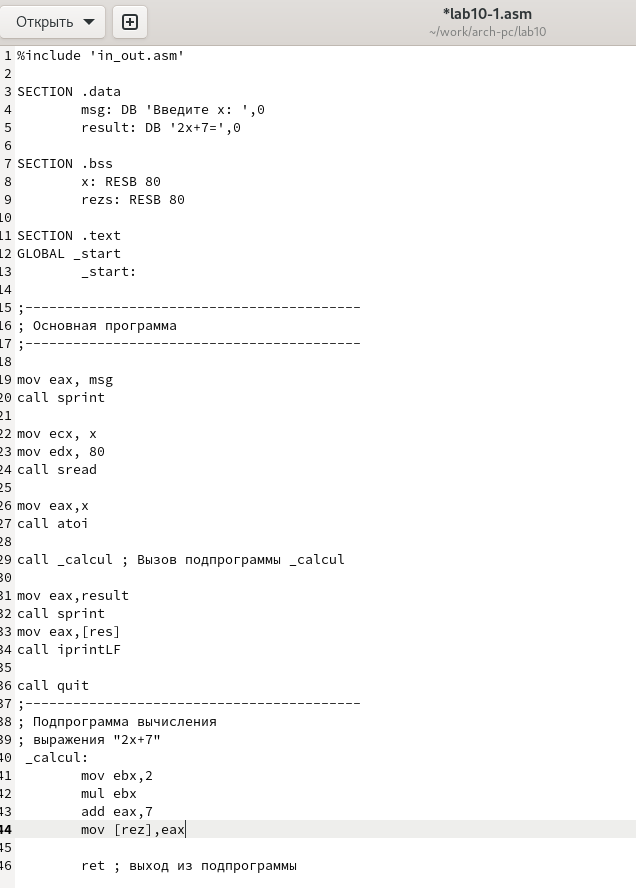


Рис. 2: Текст программы №1

1. Создаём исполняемый файл, проверяем работу. Видим ошибку из-за неправильного названия переменных (res,rez,resz).(рис. 3):

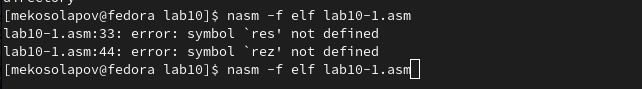


Рис. 3: Ошибка при создании исполняемого файла

1. Исправляем код программы(рис. 4):

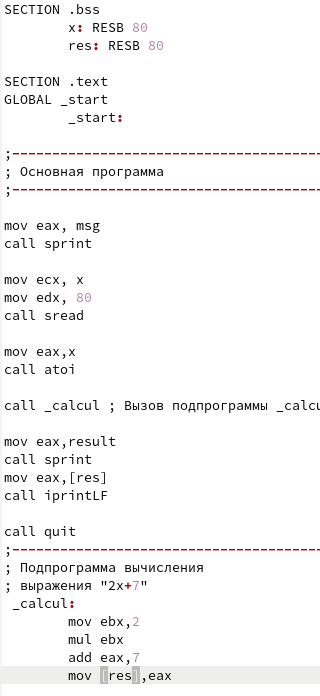


Рис. 4: Изменение текста программы

1. Создаём исполняемый файл, проверяем работу(рис. 5):

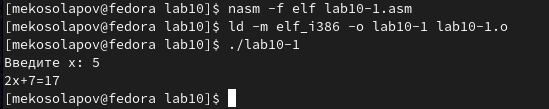


Рис. 5: Результат выполнения исправленной программы

1. Снова меняем программу, добавляя подпрограмму \*\_subcalcul\*, тоже вычисляющую функцию(рис. 6):

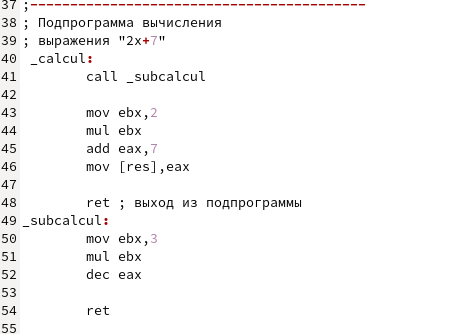


Рис. 6: Новое изменение программы. Добавление подпрограммы \_subcalcul

1. Создаём исполняемый файл, проверяем работу(рис. 7)

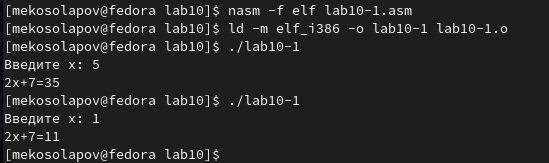


Рис. 7: Результат работы программы с подпрограммой \_subcalcul

1. Создаём файл lab10-2.asm и переносим предложенную программу из листинга №2 (рис. 8)

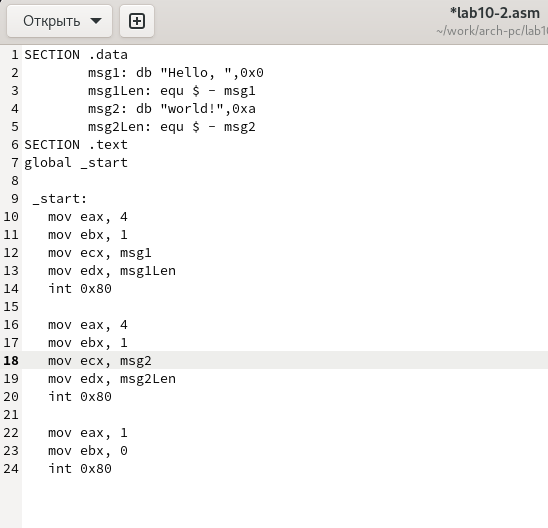


Рис. 8: Программа №2

1. Создаём исполняемый файл и файл листинга с ключом -g. Загружаем исполняемый файл в отладчик GDB(рис. 9)

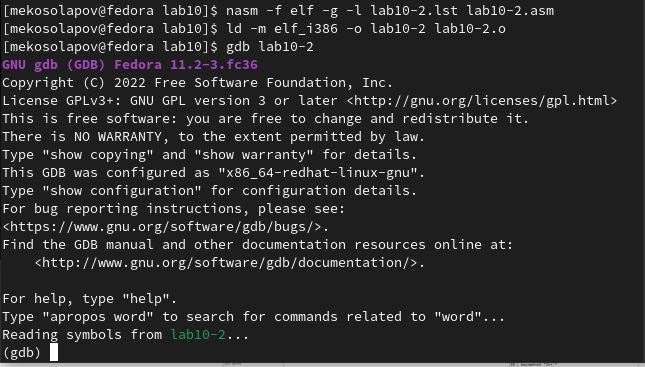


Рис. 9: Создание исполняемого файла. Его загрузка в отладчик GDB

1. Запускаем программу в оболочке GDB с помощью команды *run*(рис. 10):

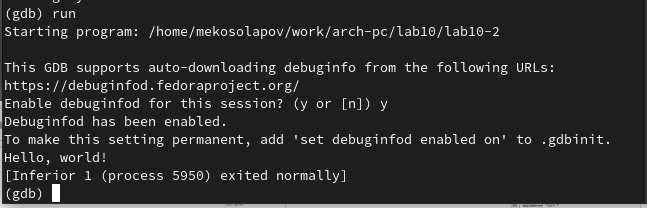


Рис. 10: Запуск программы в оболочке GDB

1. Устанавливаем брейкпоинт на метку \_start. Запускаем программу (рис. 11):

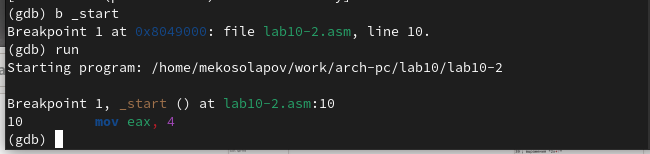


Рис. 11: Установки точки останова на метку \_start. Запуск программы

1. Смотрим дисассимилированный код программы с помощью команды *disassemble* начиная с метки \_start (рис. 12):

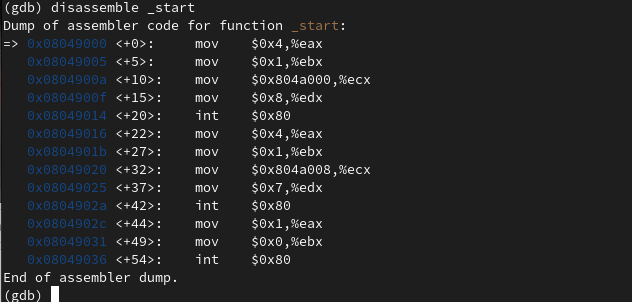


Рис. 12: Дисассимилированный код программы с помощью команды *disassemble* начиная с метки \_start

1. Переключаемся на Intel’овский синтаксис с помощью команды *set disassebly-flavor intel*. Снова смотрим дисассимилированный код. Видно что в режиме ATT сначала идут переменные, после основные регистры. А в режиме Intel наоборот (рис. 13):

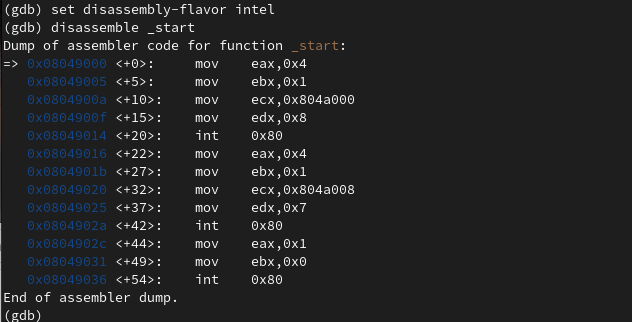


Рис. 13: Дисассимилированный код. Отображение Intel

1. Включаем режим псевдографики(рис. 14):

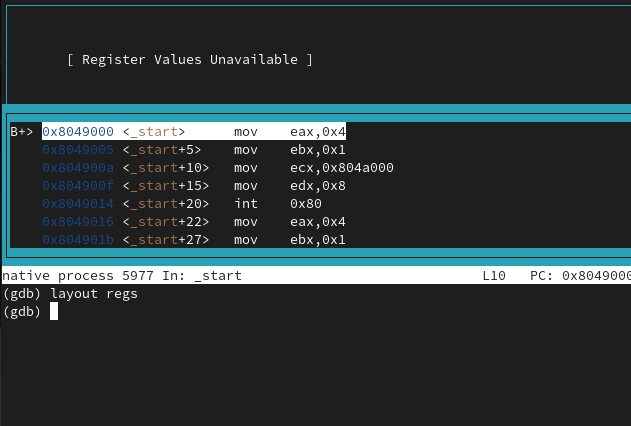


Рис. 14: Включение режим псевдографики

1. Проверяем, установлена ли точка останова с помощью команды *i b* (рис. 15):

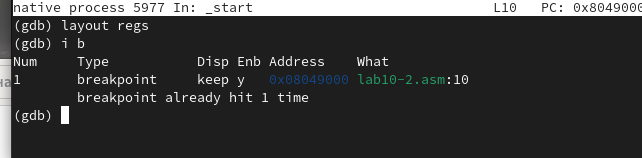


Рис. 15: Проверка установленных точек останова

1. Устанавливаем ещё одну точку останова по адресу инструкции mov eax,0x0. Смотрим все установленный breakpoints(рис. 16):

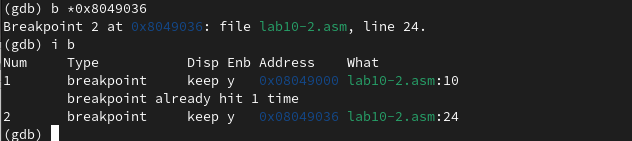


Рис. 16: Установка второй точки останова. Просмотр информации о точках останова

1. Смотрим содержание регистров с помощью команды *i r* (рис. 17):

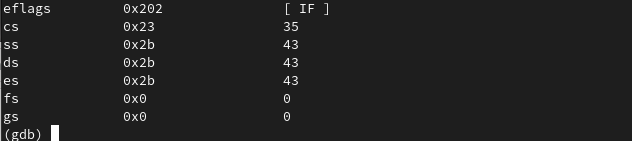


Рис. 17: Содержание регистров

1. Смотрим содержание регистров после выполнения 5 инструкций с помощью команды *si 5*. Видим, что изменились некоторые регистры. (рис. 18):

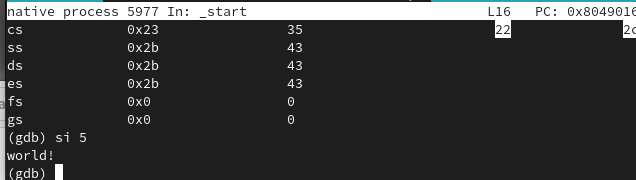


Рис. 18: Содержание регистров после выполнения команды *si 5*

1. Смотрим значение переменной msg1 по имени (рис. 19):

Рис. 19: Значение переменной msg1

Рис. 19: Значение переменной msg1

1. Смотрим значение переменной msg2 по адресу(рис. 20):

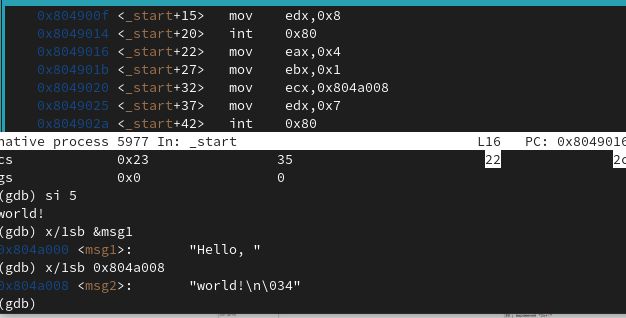


Рис. 20: Значение переменной msg2

1. Изменяем первый символ переменной msg1 с помощью команды *set*. Смотрим получившийся результат (рис. 21):

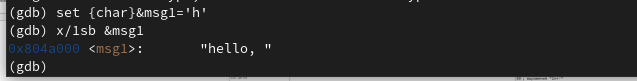


Рис. 21: Изменение первого символа строки msg1. Результат изменения

1. Так же меняем первый символ переменной msg2. Смотрим результат (рис. 22)

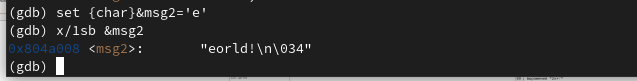


Рис. 22: Изменение первого символа строки msg2. Результат изменения

1. Выводим в шестнадцатеричном, в двоичном, в символьном форматах значение регистра edx (рис. 23)

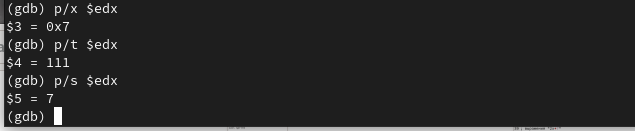


Рис. 23: Значение регистра edx в разных форматах

1. Изменяем значение регистра ebx на ‘2’ с помощью команды *set* (рис. 24)

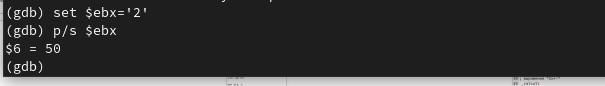


Рис. 24: Изменение значения регистра ebx на ‘2’. Результат изменения

1. Изменяем значение регистра ebx на 2 с помощью команды *set*. Видим, что выводимый результат отличается. Это потому, что в первом случае мы помещаем в регистр строку ‘2’, а во втором число 2(рис. 25):

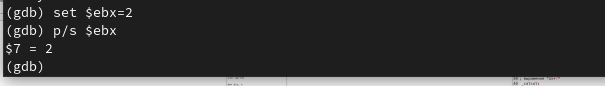


Рис. 25: Изменение значения регистра ebx на 2. Результат изменения

1. Завершаем выполнение программы с помощью команды *c* (рис. 26):



Рис. 26: Завершение программы и выход из GDB

1. Копируем файл lab9-2.asm в файл с именем lab10-3.asm (рис. 27):

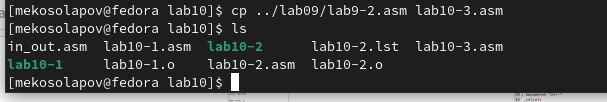


Рис. 27: Копирование файла lab9-2.asm в файл с именем lab10-3.asm

1. Создаём исполняемый файл и файл листинга с ключом -g (рис. 28):

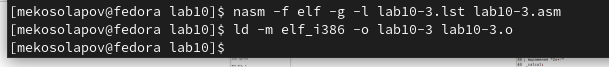


Рис. 28: Создание исполняемого файла, файла листинга lab10-3

1. Загружаем исполняемый файл в отладчик GDB, указывая аргументы (рис. 29):

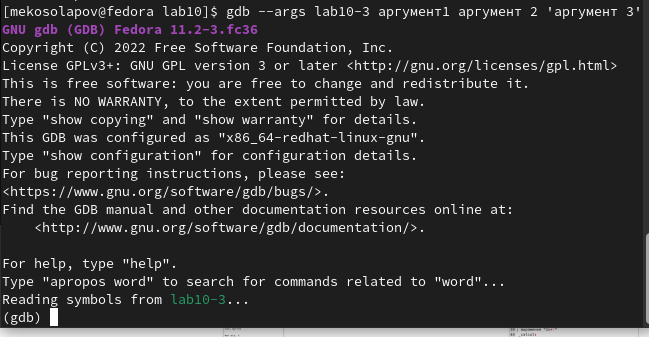


Рис. 29: Загрузка исполняемого файла в отладчик gdb с аргументами

1. Устанавливаем точку останова перед \_start и запускаем программу(рис. 30):

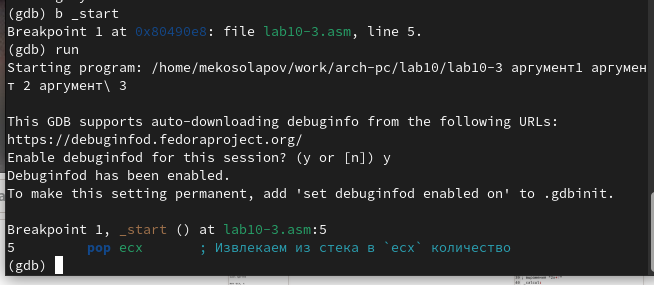


Рис. 30: Установка точки останова перед меткой \_start. Запуск программы

1. Узнаём количеству аргументов командной строки (рис. 31):

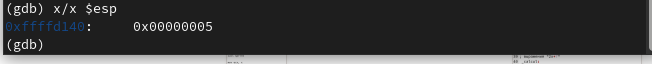


Рис. 31: Количество аргументов командной строки

1. Смотрим позиции стека по адресам [esp+4n], n={1,2,3,4,5,6}. Шаг равен 4, потому что на каждый аргумент выделено 4 байта (рис. 32):

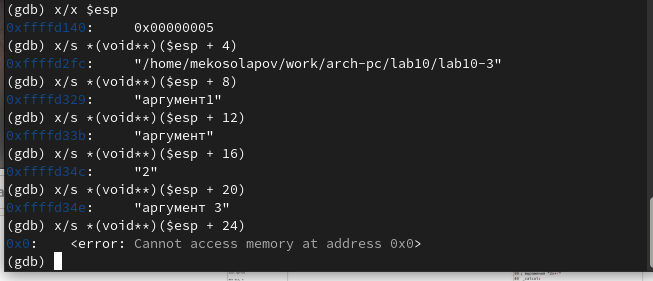


Рис. 32: Значения располагающиеся в стеке

***Задания для самостоятельной работ*** 33. Скопируем программу из лабораторной работы №9 и реализуем в ней вычисление функции через подпрограмму (рис. 33):

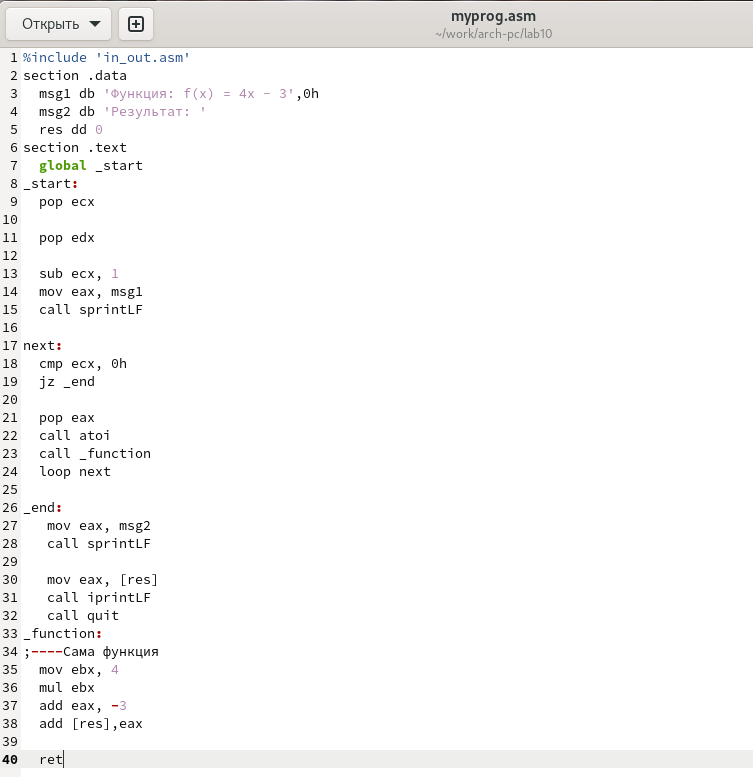


Рис. 33: Код программы

1. Создаём исполняемый файл и проверяем его работу (рис. 34):

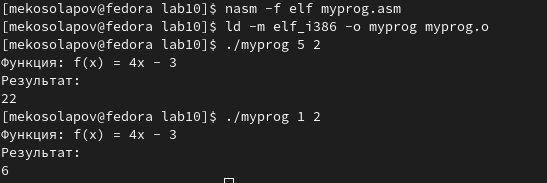


Рис. 34: Создание исполняемого файла, проверка его работы

1. Теперь найдём ошибку в предложенной программе и исправим её. С помощью gdb отслеживаем работу программы. Исправляем программу для вычисления умножения, поменяв регистры местами (сначала eax умножался на 4, а не ebx) (рис. 35):

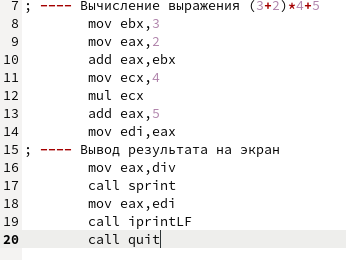


Рис. 35: Исправление ошибки

1. Создаём исполняемый файл и проверяем его работу (рис. 36):

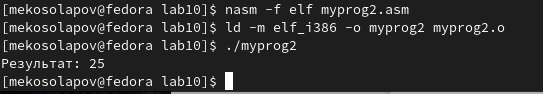


Рис. 36: Создание исполняемого файла, проверка его работы

# 5 Выводы

В ходе данной лабораторной работы я научился работать с программой отладки кода, приобрёл навыки написания программ с использованием подпрограмм.

# Список литературы