Отчёт по лабораторной работы №9

Дисциплина: архитектура компьютера

Худдыева Дженнет

Содержание

# 1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм.Знакомство с методами отладки при помощи GDB и его основны- ми возможностьями.

# 2 Задание

1.Реализация подпрограмм в NASM

2.Отладки подпрограмм с помощью GDB.

3.Добавление точек останова.

4.Работа с данными подпрограмм в GDB.

5.Обработка аргументов командной строки в GDB.

6.Задания для самостоятельной работы.

# 3 Теоретическое введение

Отладка - это процесс поиска и исправления ошибок в программе.Отладчики позволяют управлять ходом выполнения подпрограммы, контролировать и изменять данные.Это помогает быстрее найти место ошибки в программе и ускорить её исправления.Наиболее популрные способы работы с отладчиком - это использование точек останова и выполнение программы по шагам.

GDB(GNU Debugger -отладчик проекта GNU) работает на многих UNIX - подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ.Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли.Однако для GDB существует несколько сторонних графических надстроек,а кроме того,некоторые интегрированниые среды разработки используют его в качестве базовой подсистемы отладки.

Отладчик GDB(как и любой другой отладчик) позволяет увидеть,что происходит “внутри” программы в момент её выполнения или что делает программа в момент сбоя.

Команда run-запускает отлаживаемую программу в оболочке GDB.

Команда kill-прекращает отладку программы,после чего следует вопрос о прекращении процесса отладки.Если в ответ введено y(да), отладка программы прекращается.Командой run её можно начать заново,при этом все точки останова(breakpoints), точки просмотра (watchpoints) и точка отлова(catchpoints) сохраняются.

Для выхода из отладчика используется команда quit(q).

Если есть файл с исходным текстом программы,а в исполняемый файл включена информация о номерах строк исходного кода,то программу можно отлаживать,работая в отладчике непосредственно с её исходным текстом.Чтобы программу иожно было отлаживать на уровне строк исходного кода,она должна быть откомпилирована с ключом -g.

Установить точку останова можно командой break.Типичный аргумент этой команды-место установки.Его можно задать как имя метки, или как адрес.Чтобы не было путаницы с номерами,перед адресом ставится \*

Команда si позволяет выполнять программу по шагам.

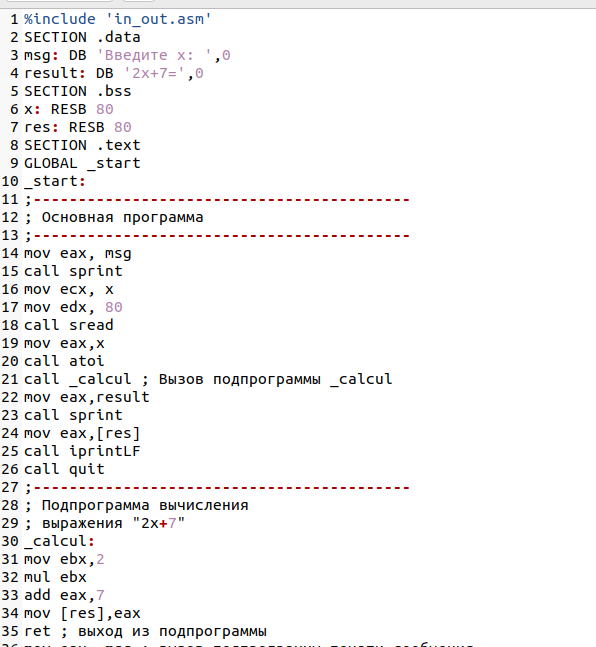
Подпрограмма-это как правило,функционально законченный участок кода,который можно многократно вызывать из разных мест программы В отличие от простых переходов из подпрограмм существует возврат на команду,следующую за вызовом.Если в программе встречается одинаковый участок кода,его можно оформить в виде подпрограммы,а во всех нужных местах поставить её вызов.При этом подпрограмма будет содержаться в коде в одном экземпляре,что позволяет уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция call, которая заносит адрес следующей инструкции в стек и загружает в регистр eip адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией ret, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией call, и заносит его в eip. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией call.

# 4 Выполнение лабораторной работы

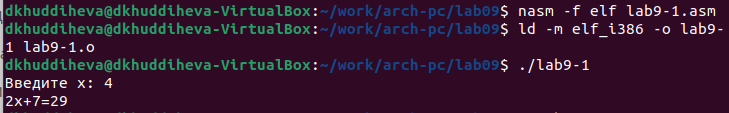
## 4.1 **Реализация подпрограмм в NASM**

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab9-1.asm. Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис.[??])



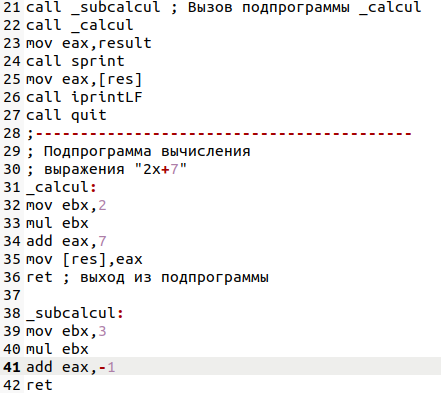
Создание файлов для лабораторной работы

Создаю исполняемый файл и проверяю его работу. (рис. [??])



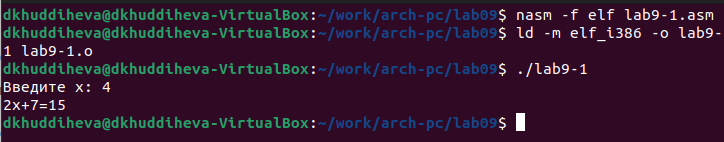
Запуск исполняемого файла

Изменяю 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. [??])



Изменение текста программы

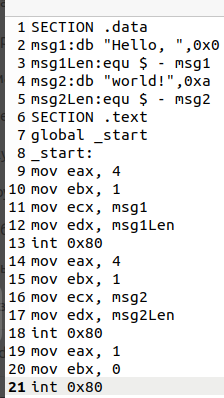
Создаю исполняемый файл и проверяю его работу. (рис. [??])



Запуск исполняемого файла

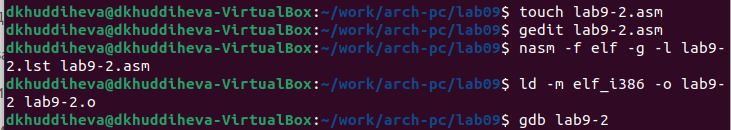
## 4.2 **Отладка программам с помощью GDB**

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (рис. [??])



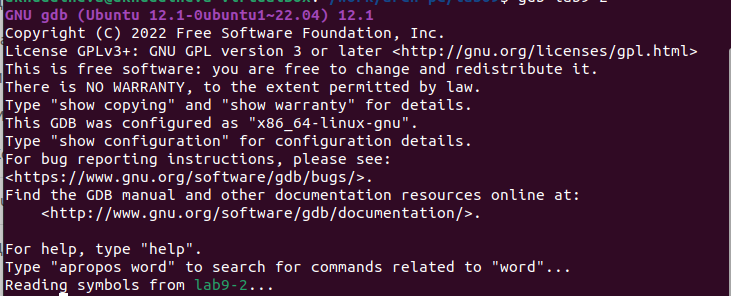
Ввод текста программы

Получаю исполняемый файл для работы с GDB с ключом ‘-g’. (рис. [??])



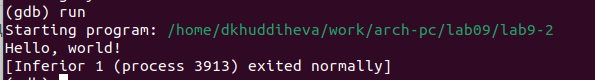
Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb. (рис. [??])



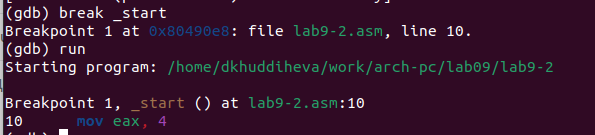
Загрузка исполняемого файла в отладчик

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



Проверка работы файла с помощью команды run

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start и запускаю её. (рис.[??])



Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы с помощью команды disassemble, начиная с метки \_start, и переключаюсь на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel. (рис. [??])



Использование команд disassemble и disassembly-flavor intel



Использование команд disassemble и disassembly-flavor intel

В режиме ATT имена регистров начинаются с символа %, а имена операндов с $, в то время как в Intel используется привычный нам синтаксис.

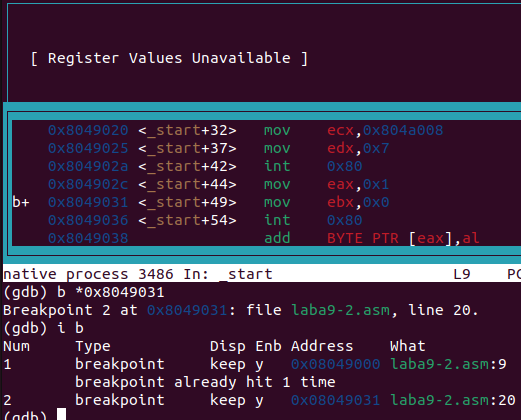
Включаю режим псевдографики для более удобного анализа программы с помощью команд layout asm и layout regs. (рис. [??])



Включение режима псевдографики

### 4.2.1 **Добавление точек останова**

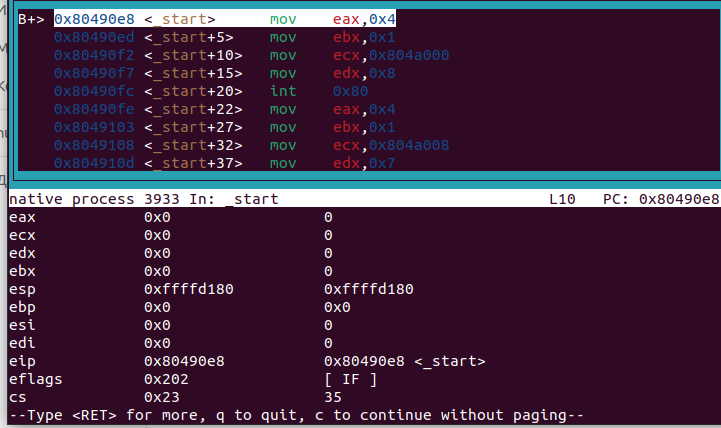
Проверяю, что точка останова по имени метки \_start установлена с помощью команды info breakpoints и устанавливаю еще одну точку останова по адресу инструкции mov ebx,0x0. Просматриваю информацию о всех установленных точках останова. (рис. [??])



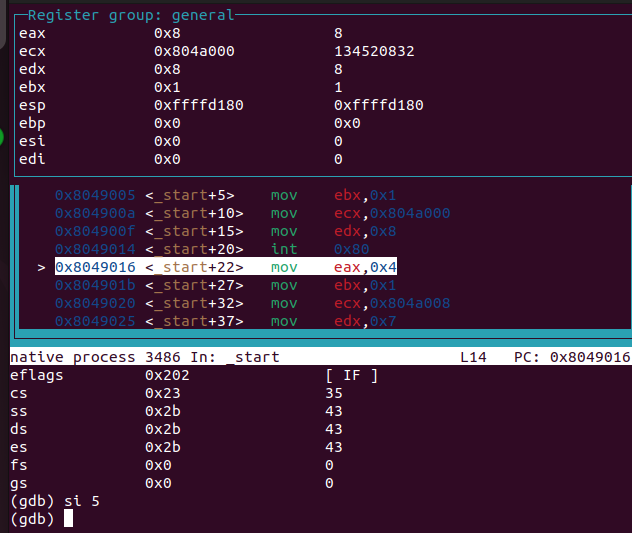
Установление точек останова и просмотр информации о них

### 4.2.2 **Работа с данными программы в GDB**

Выполняю 4 инструкций с помощью команды stepi и слежу за изменением значений регистров. (рис. [??])



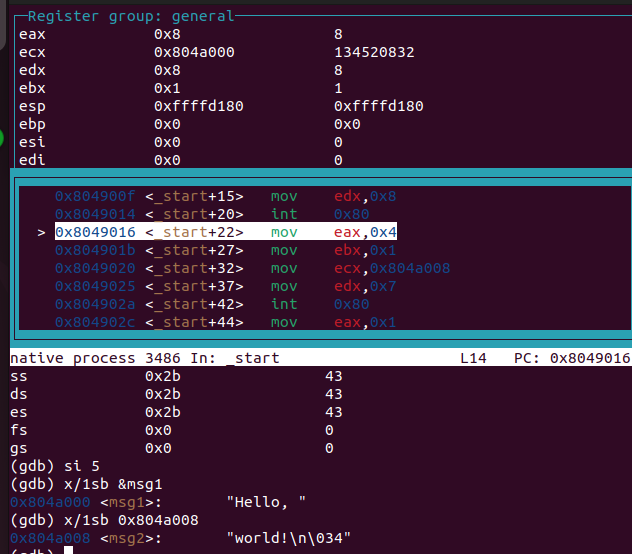
До использования команды stepi



До использования команды stepi

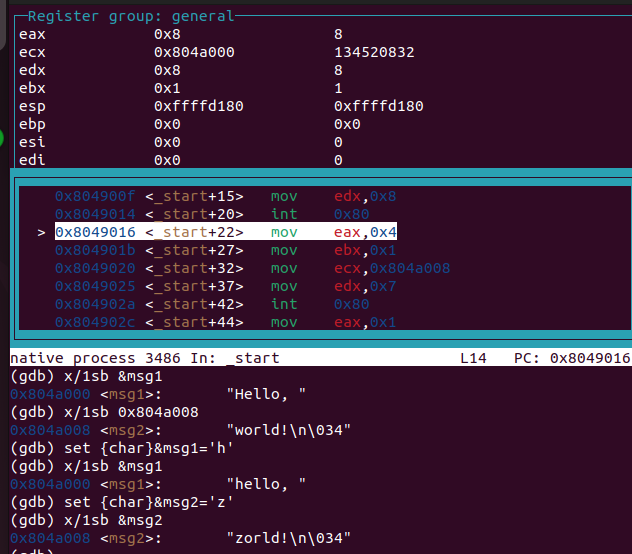
Изменились значения регистров eax, ecx, edx и ebx.

Просматриваю значение переменной msg1 с помощью команды x/1sb &msg1значение переменной msg2 по ее адресу. (рис. [??])



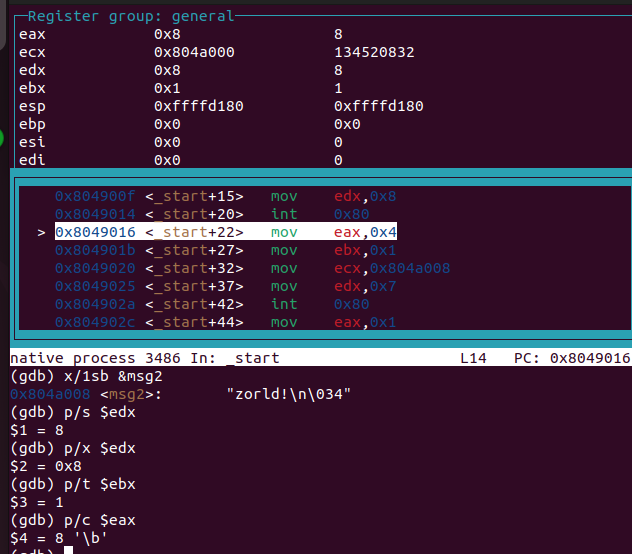
Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2. (рис. [??])



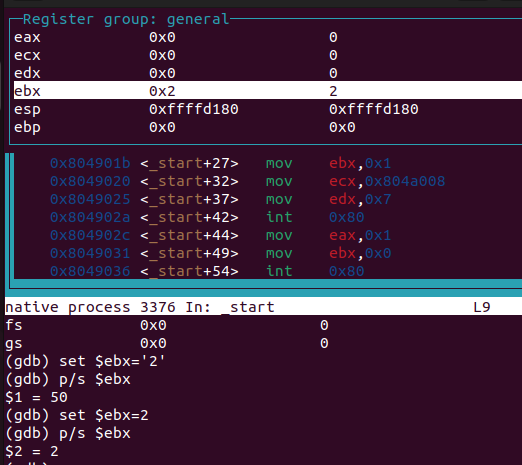
Использование команды set

Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра edx с помощью команды print p/F $val. (рис. [??])



Вывод значения регистра в разных представлениях

С помощью команды set изменяю значение регистра ebx в соответствии с заданием. (рис. [??])



Использование команды set для изменения значения регистра

Разница вывода команд p/s $ebx отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется.

Завершаю выполнение программы с помощью команды continue и выхожу из GDB с помощью команды quit.

### 4.2.3 **Обработка аргументов командной строки в GDB**

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создаю исполняемый файл. (рис. [??])

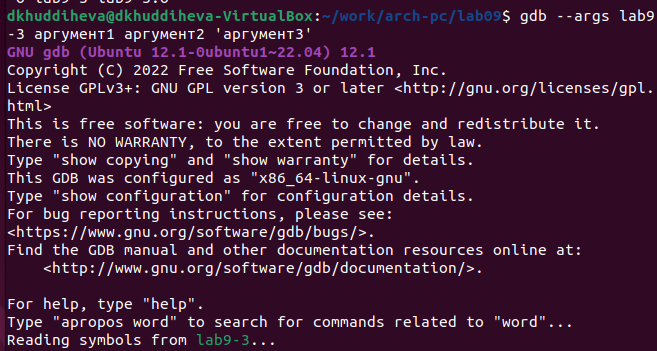
Создание файла

Создание файла

Создание файла

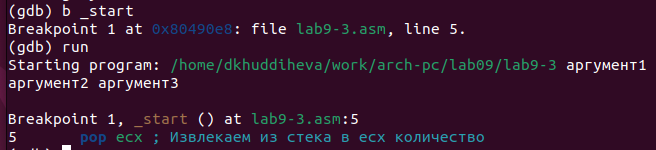
Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа –args. (рис. [??])



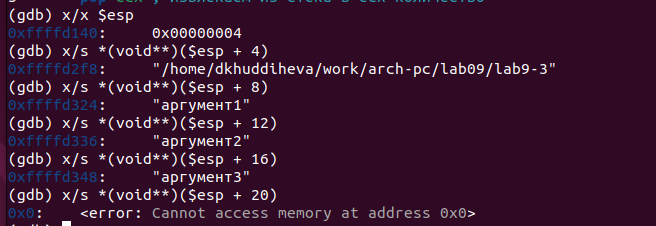
Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. [??])



Установление точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам. (рис.[??])

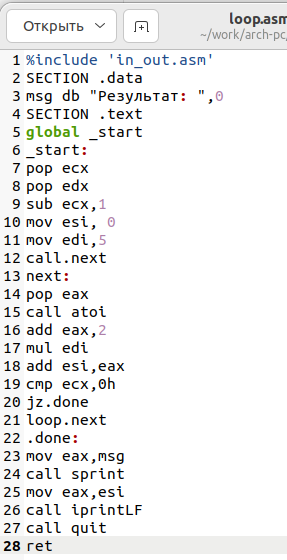


Просмотр значений, введенных в стек

Шаг изменения адреса равен 4, т.к количество аргументов командной строки равно 4.

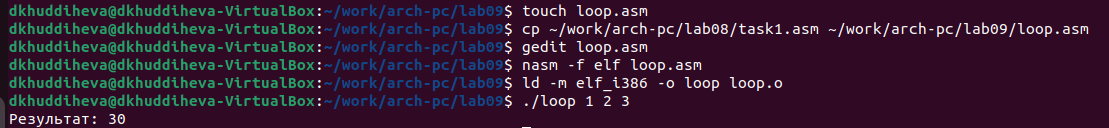
## 4.3 **Задания для самостоятельной работы**

1. Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму. (рис. [??])



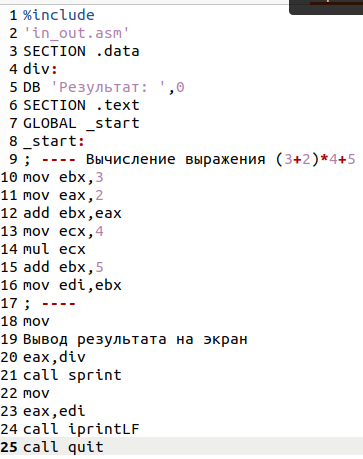
Написание кода подпрограммы

Запускаю код и проверяю, что она работает корректно. (рис. [??])



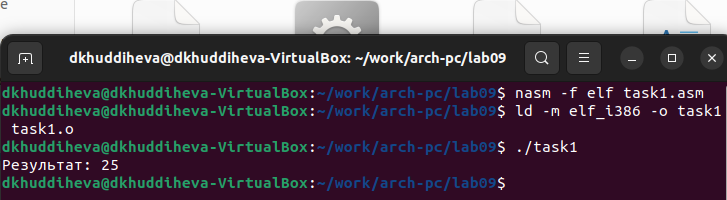
Запуск программы и проверка его вывода

1. Ввожу в файл task1.asm текст программы из листинга 9.3. (рис. [??])



Ввод текста программы из листинга 9.3

При корректной работе программы должно выводится “25”. Создаю исполняемый файл и запускаю его. (рис. [??])



Создание и запуск исполняемого файла

# 5 Выводы

Во время выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.