Лабораторная работа №9

НКАбд-06-23

Улитина Мария Максимовна

Содержание

[1 Цель работы 1](#__RefHeading___Toc631_774103658)

[2 Задание 2](#__RefHeading___Toc633_774103658)

[3 Теоретическое введение 2](#__RefHeading___Toc635_774103658)

[3.1 Понятие об отладке 2](#__RefHeading___Toc637_774103658)

[3.2 Методы отладки 3](#__RefHeading___Toc639_774103658)

[3.3 Основные возможности отладчика GDB 3](#__RefHeading___Toc641_774103658)

[3.4 Запуск отладчика GDB; выполнение программы; выход 4](#__RefHeading___Toc643_774103658)

[3.5 Дизассемблирование программы 4](#__RefHeading___Toc645_774103658)

[3.6 Точки останова 4](#__RefHeading___Toc647_774103658)

[3.7 Пошаговая отладка 5](#__RefHeading___Toc649_774103658)

[3.8 Работа с данными программы в GDB 5](#__RefHeading___Toc651_774103658)

[3.9 Понятие подпрограммы 5](#__RefHeading___Toc653_774103658)

[3.9.1 Инструкция call и инструкция ret 5](#__RefHeading___Toc655_774103658)

[4 Выполнение лабораторной работы 6](#__RefHeading___Toc657_774103658)

[4.1 Задания для самостоятельной работы 12](#__RefHeading___Toc659_774103658)

[5 Выводы 14](#__RefHeading___Toc661_774103658)

[Список литературы 14](#__RefHeading___Toc663_774103658)

# 1 Цель работы

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

# 2 Задание

1. Реализация подпрограмм в NASM.
2. Отладка программ с помощью GDB.
3. Добавление точек останова.
4. Работа с данными программы в GDB.
5. Обработка аргументов командной строки в GDB.
6. Преобразование программы из лабораторной работы №8, реализовав вычисление значения функции как подпрограмму.
7. Проверить неправильную работу программы, проанализировав изменения значения регистров. Определить ошибку и исправить ее.

# 3 Теоретическое введение

## 3.1 Понятие об отладке

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

• обнаружение ошибки;

• поиск её местонахождения;

• определение причины ошибки;

• исправление ошибки.

Можно выделить следующие типы ошибок:

• синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка;

• семантические ошибки — являются логическими и приводят к тому, что программа запускается, отрабатывает, но не даёт желаемого результата;

• ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга. Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

## 3.2 Методы отладки

Наиболее часто применяют следующие методы отладки:

• создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);

• использование специальных программ-отладчиков.

Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам. Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить другие действия. Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова:

• Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);

• Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

## 3.3 Основные возможности отладчика GDB

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 интегрированные 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. GDB может выполнять следующие действия:

• начать выполнение программы, задав всё, что может повлиять на её поведение;

• остановить программу при указанных условиях;

• исследовать, что случилось, когда программа остановилась;

• изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

## 3.4 Запуск отладчика GDB; выполнение программы; выход

Синтаксис команды для запуска отладчика имеет следующий вид: gdb *[опции] [имя\_ \_файла | ID процесса]* После запуска gdb выводит текстовое сообщение — так называемое «nice GDB logo». В следующей строке появляется приглашение (gdb) для ввода команд.

## 3.5 Дизассемблирование программы

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 можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом -g.

## 3.6 Точки останова

Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»: (gdb) break \* (gdb) b Информацию о всех установленных точках останова можно вывести командой info (кратко i): (gdb) info breakpoints (gdb) i b Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой disable: disable breakpoint Обратно точка останова активируется командой enable: enable breakpoint Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды delete: (gdb) delete breakpoint Ввод этой команды без аргумента удалит все точки останова. Информацию о командах этого раздела можно получить, введя help breakpoints

## 3.7 Пошаговая отладка

Для продолжения остановленной программы используется команда continue (c) (gdb) с *[аргумент]*. Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число 𝑁, которое указывает отладчику проигнорировать 𝑁 − 1 точку останова (выполнение остановится на 𝑁-й точке). Команда stepi (кратко sI) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию: *(gdb) si [аргумент]* При указании в качестве аргумента целого числа 𝑁 отладчик выполнит команду step 𝑁 раз при условии, что не будет точек останова или выполнение программы не прервётся по другим причинам. Команда nexti (или ni) аналогична stepi, но вызов процедуры (функции) трактуется отладчиком как одна инструкция: *(gdb) ni [аргумент]* Информацию о командах этого раздела можно получить, введя (gdb) help running

## 3.8 Работа с данными программы в GDB

Как уже упоминалось, отладчик может показывать содержимое ячеек памяти и регистров, та при необходимости позволяет вручную изменять значения регистров и переменных.Посмотреть содержимое регистров можно с помощью команды info registers (или i r): (gdb) info registers

## 3.9 Понятие подпрограммы

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 позволит уменьшить размер кода всей программы.

### 3.9.1 Инструкция call и инструкция ret

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

Создадим каталог и файл для лабораторной работы (рис. [1](#fig:001)).

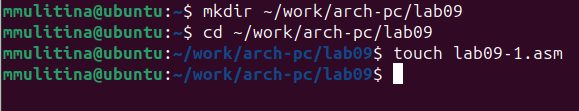


Figure 1: Создание каталога

Введём текст программы и запустим её для проверки (рис. [2](#fig:002)).

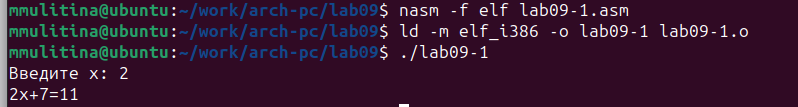


Figure 2: Программа

Добавим подпрограмму \_subcalcul, запустим программу для проверки (рис. [3](#fig:003)).

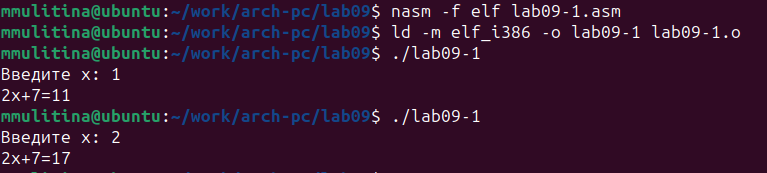


Figure 3: Программа

Создадим файл lab09-2.asm, введём в него текст программы, получим исполняемый файл и загрузим его в отладчик (рис. [4](#fig:004)).

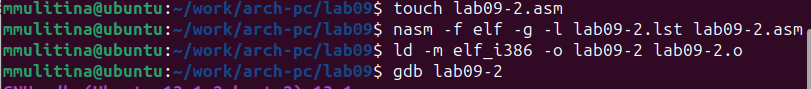


Figure 4: Отладчик

Проверим работу программы, запустим ее в оболочке GDB (рис. [5](#fig:005)).

Figure 5: Отладчик

Figure 5: Отладчик

(рис. [6](#fig:006)).

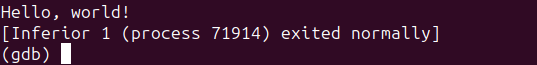


Figure 6: Отладчик

Установим брейкпоинт и запустим программу (рис. [7](#fig:007)).

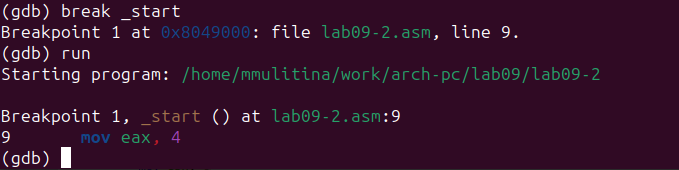


Figure 7: Брейкпоинт

Посмотрим дисассимилированный код (рис. [8](#fig:008)).



Figure 8: Дисассимилированный код

Переключимся на отображение команд с Intel синтаксисом (рис. [9](#fig:009)).



Figure 9: Отображение

Включим режим псевдографики (рис. [10](#fig:010)).

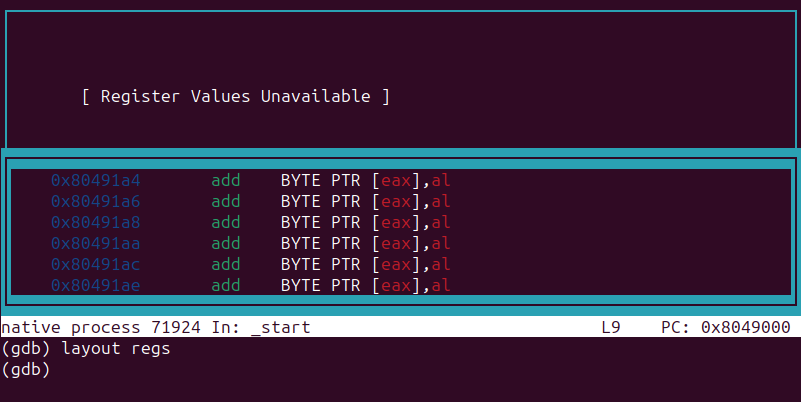


Figure 10: Псевдографика

Проверим точки останова с помощью команды info breakpoints (рис. [11](#fig:011)).

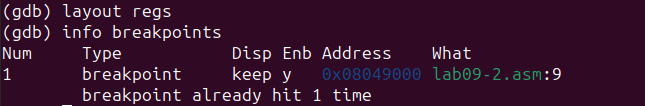


Figure 11: Точки останова

Установим ещё одну точку останова и снова посмотрим информацию о точках останова (рис. [12](#fig:012)).

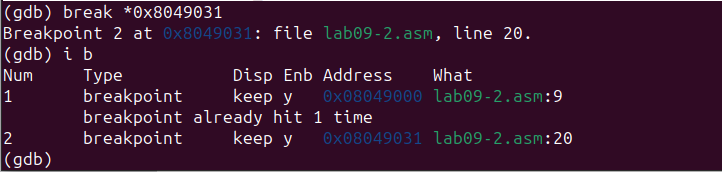


Figure 12: Точки останова

Выполним 5 инструкций с помощью команды stepi и проследим изменения регистров (рис. [13](#fig:013)).

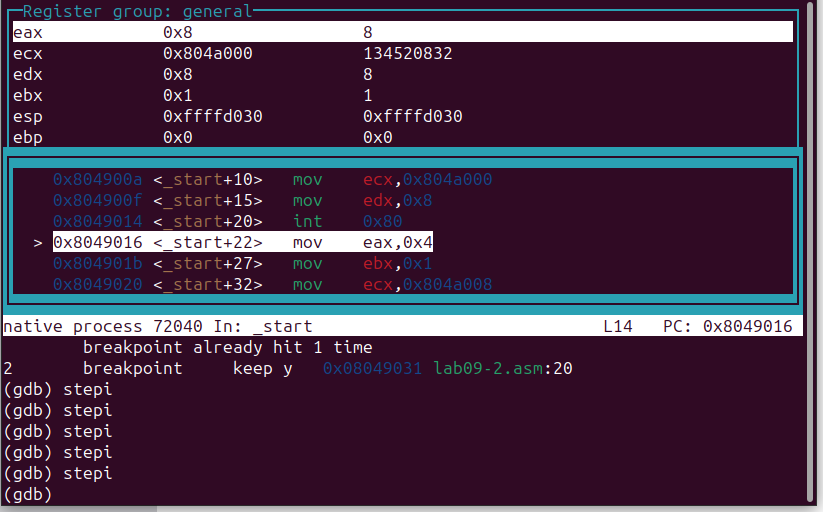


Figure 13: stepi

Изменились значения регистров eax, ecx, ebx, edx.

Посмотрим значения регистров с помощью info registers (рис. [14](#fig:014)).

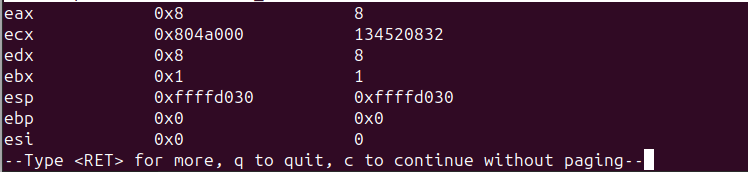


Figure 14: Значения регистров

Посмотрим значение переменной msg1 по имени (рис. [15](#fig:015)).

Figure 15: Значение переменной

Figure 15: Значение переменной

Изменим первый символ переменной (рис. [16](#fig:016)).

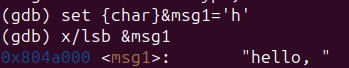


Figure 16: Изменение символа

Изменим любой символ второй переменной msg2 (рис. [17](#fig:017)).

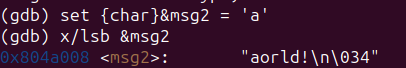


Figure 17: Изменение символа

Посмотрим значений регистра edx (рис. [18](#fig:018)).

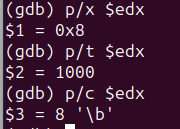


Figure 18: Значения регистра

С помощью set изменим значение регистра ebx (рис. [19](#fig:019)).

Figure 19: Изменение значения регистра

Figure 19: Изменение значения регистра

и проверим его значение (рис. [20](#fig:020)).

Figure 20: Значение

Figure 20: Значение

Снова изменим значение ebx (рис. [21](#fig:021)).

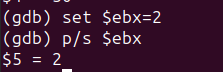


Figure 21: Значение

В первом случае мы ввели символьное значение, во втором цифру.

Скопируем файл из прошлой лабораторной работы (рис. [22](#fig:022)).

Figure 22: Копирование файла

Figure 22: Копирование файла

Создадим исполняемый файл (рис. [23](#fig:023)).

Figure 23: Исполняемый файл

Figure 23: Исполняемый файл

Загрузим исполняемый файл в отладчик, указав аргументы (рис. [24](#fig:024)).

Figure 24: Отладчик

Figure 24: Отладчик

Установим точку останова перед первой инструкцией в программе и запустим её (рис. [25](#fig:025)).

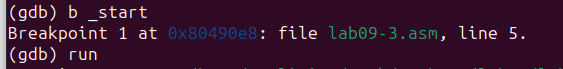


Figure 25: Точка останова

Посмотрим значение регистра esp, где хранится адрес вершины стека (рис. [26](#fig:026)).

Figure 26: Регистр esp

Figure 26: Регистр esp

Посмотрим остальные позиции стека по адресу (рис. [27](#fig:027)).

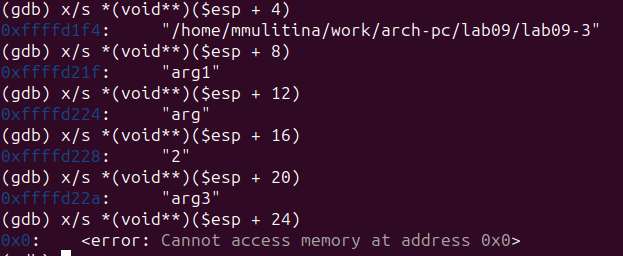


Figure 27: Позиции стека

Шаг изменения равен 4, т.к. у нас 4 аргумента.

## 4.1 Задания для самостоятельной работы

1. Преобразуем программу из лабораторной работы №8, реализовав вычисление значения функции как подпрограмму (рис. [28](#fig:028)).

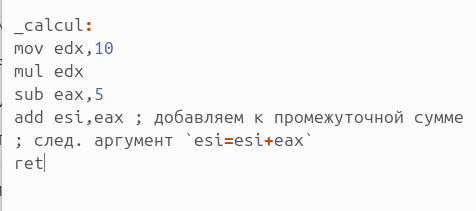


Figure 28: Подпрограмма

Запустим программу для проверки (рис. [29](#fig:029)).

Figure 29: Проверка

Figure 29: Проверка

1. Создадим файл для программы, введём в него текст программы, запустим его в отладчике GDB (рис. [30](#fig:030)).

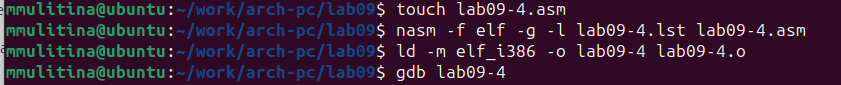


Figure 30: Файл

При умножении с помощью mul, мы умножаем eax на ecx b записываем в eax. Получаем 2*4=9 вместо (3+2)*4 (рис. [31](#fig:031)).

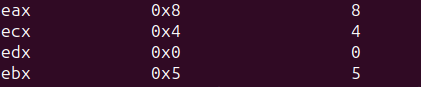


Figure 31: Проверка

Потом скалдываем с регистром ebx 5 и получаем 10. Проверим это, запустив программу (рис. [32](#fig:032)).

Figure 32: Запуск программы

Figure 32: Запуск программы

Исправим программу (рис. [33](#fig:033)).

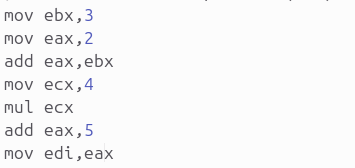


Figure 33: Исправление программы

Запустим её для проверки (рис. [34](#fig:034)).

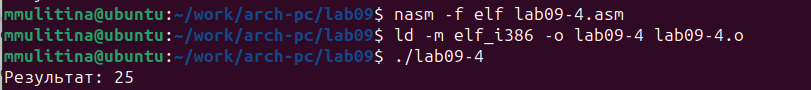


Figure 34: Запуск

# 5 Выводы

В процессе выполнения работы я приобрела навыки написания программ с использованием подпрограмм и познакомилась с методами отладки при помощи GDB и его основными возможностями.

# Список литературы

Лабораторная работа №9.