Отчёт по лабораторной работе №9

дисциплина: архитектура компьютера

Терещенкова Маргарита Владимировна

Содержание

# 1 Цель работы

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

# 2 Задание

1. Релизация 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. Задание для самостоятельной работы.

# 3 Теоретическое введение

## 3.1 Понятие об откладке

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.

## 3.2 Методы откладки

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.

# 4 Выполнение лабораторной работы

## 4.1 Реализация подпрограмм в NASM

1. Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm:

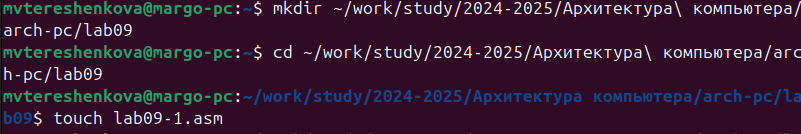


Рис. 1: Создание файла

1. Ввожу в файл lab09-1.asm текст программы из листинга 9.1.

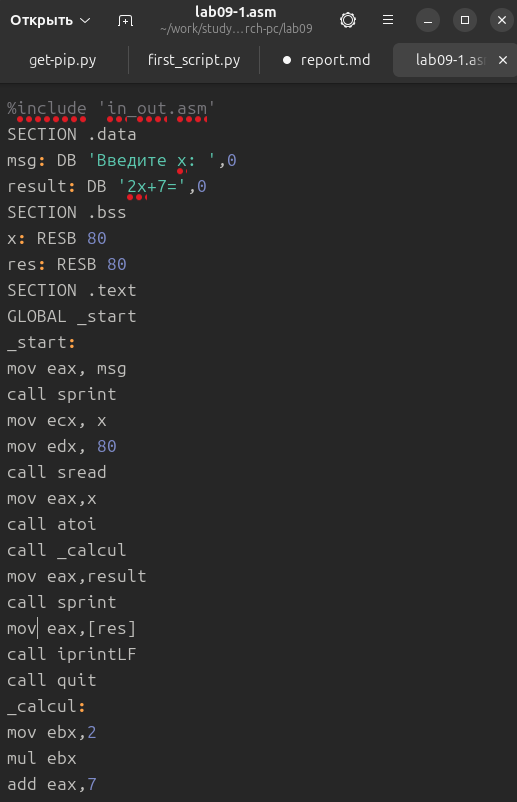


Рис. 2: Редактирование файла

Создаю исполняемый файл и запускаю его.

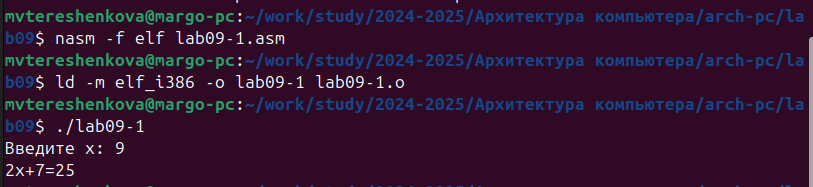


Рис. 3: Запуск файла

Программа работает корректно.

## 4.2 Отладка программам с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2.

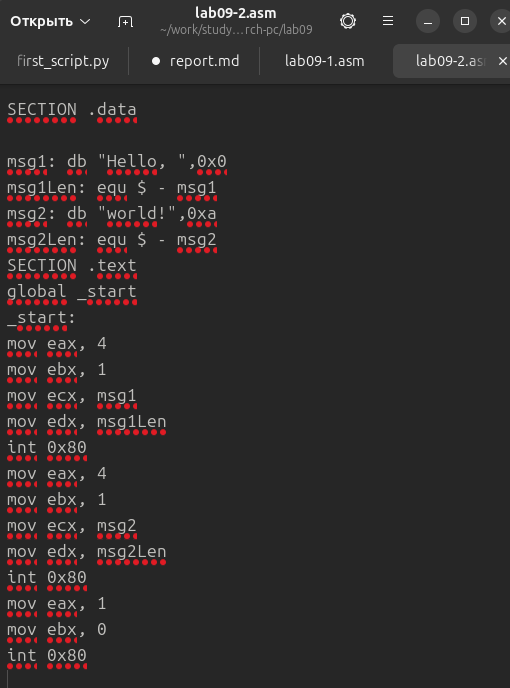


Рис. 4: Редактирование файла

Загружаю исполняемый файл в отладчик gdb. Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run.

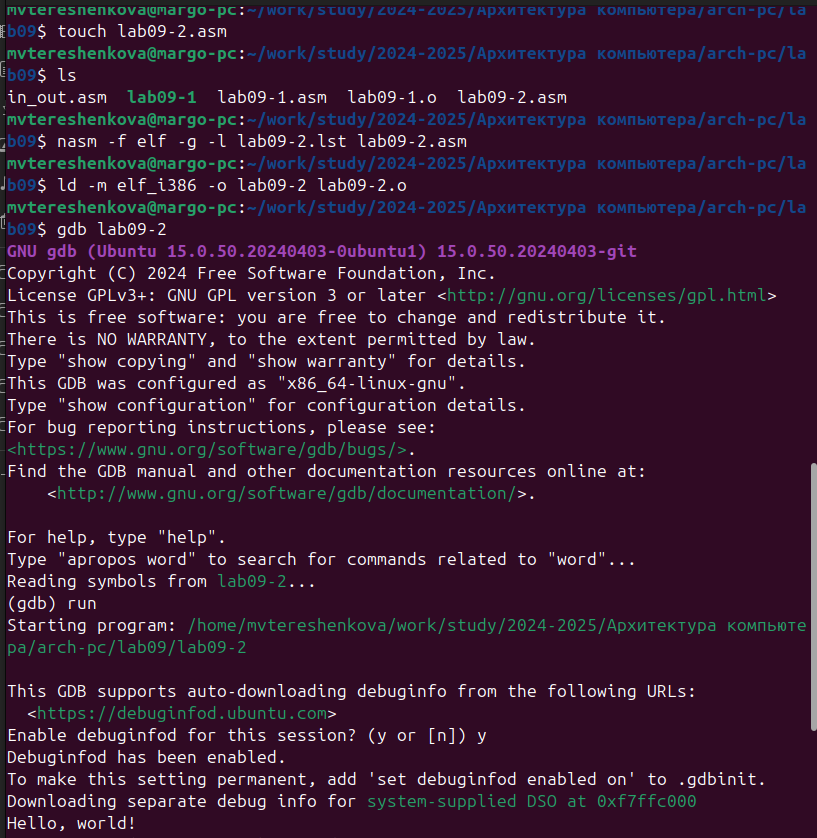


Рис. 5: Запуск файла

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаю её.

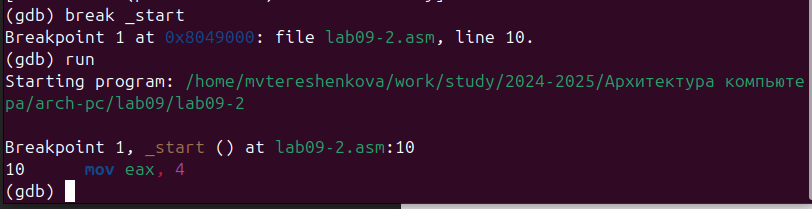


Рис. 6: Брейкпоинт на метку \_start

Посмотрела дисассимилированный код программы с помощью команды disassemble, начиная с метки \_start.

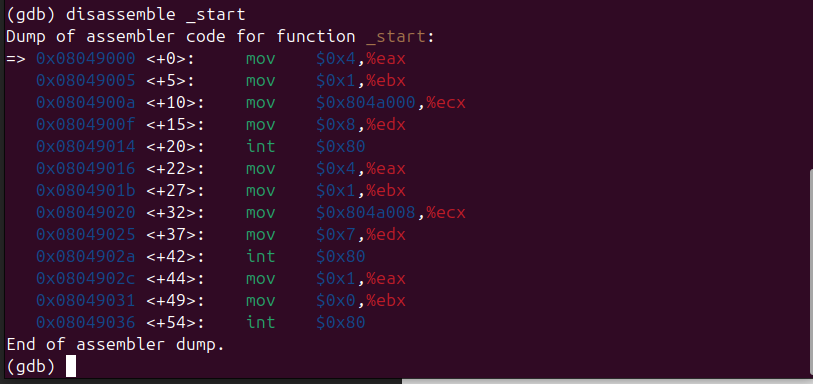


Рис. 7: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel.



Рис. 8: Код программы

Включаю режим псевдографики для более удобного анализа программы.

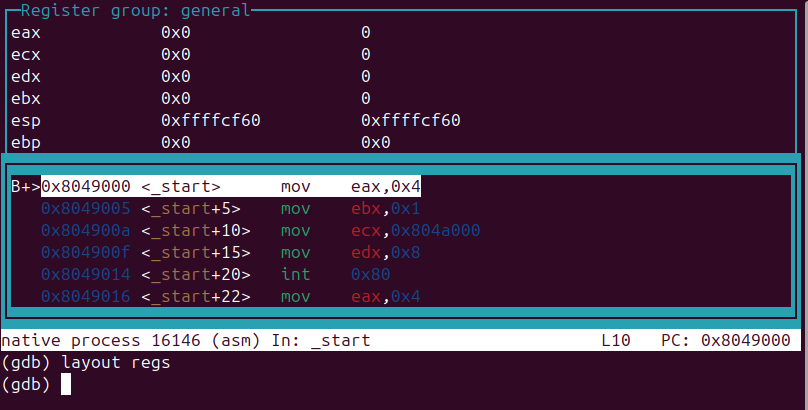


Рис. 9: Режим псевдографики

## 4.3 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (\_start). Про- веряю это с помощью команды info breakpoints (кратко i b).

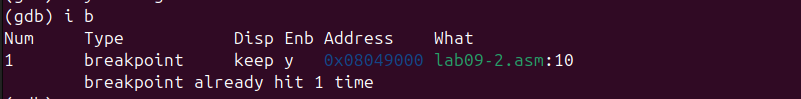


Рис. 10: Проверка точки останова

Точка установлена.

Установила еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определяю адрес предпоследней инструкции (mov ebx,0x0) и установливаю точку останова. Посмотрела информацию о всех установленных точках останова.

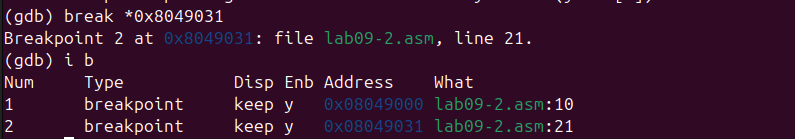


Рис. 11: Установка и проверка точки останова

## 4.4 Работа с данными программы в GDB

Посмотрела содержимое регистров с помощью команды info registers.

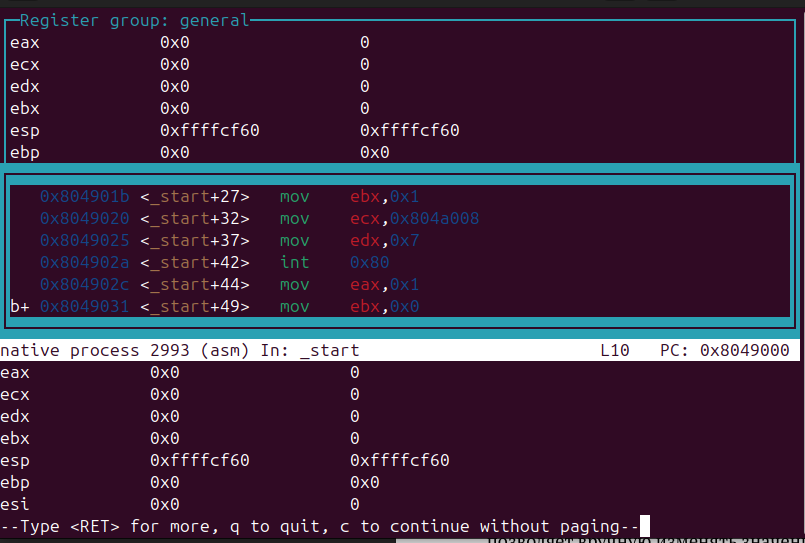


Рис. 12: Просмотр содержимого регистров

Посмотрела значение переменной msg1 по имени.

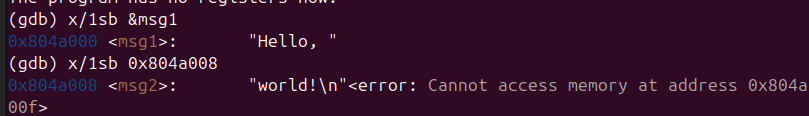


Рис. 13: Просмотр значения переменной msg1 и msg2

Изменила значение для регистра или ячейки памяти с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. Изменила первый символ переменной msg1 “H” на “h”.

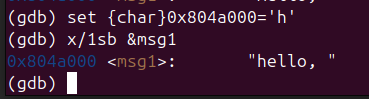


Рис. 14: Изменение первого символа с помощью команды set

Заменила первый символ во второй переменной msg2 (“w” на “g”).

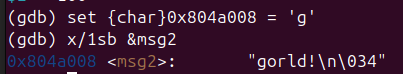


Рис. 15: Изменение первого символа с помощью команды set

Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.

* **/x - шестнадцатеричная**
* **/t - двоичная**
* **/c - символьный**

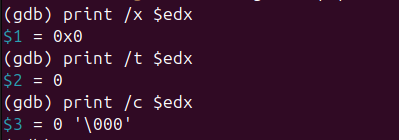


Рис. 16: Различные форматы значения регистра edx

С помощью команды set изменила значение регистра ebx.

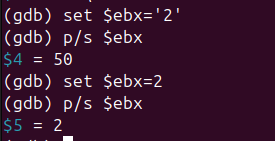


Рис. 17: Изменение значения регистра ebx

Разница в том, что при установке **ebx=2**, значение напрямую интерпретируется как число, которое GDB не может отобразить как строку, и просто возвращает его.

Завершаю выполнение программы с помощью команды stepi (сокращенно si) и вышла из GDB с помощью команды quit (сокращенно q).

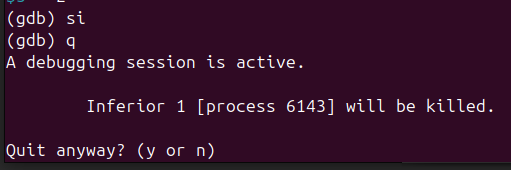


Рис. 18: Завершение выполнение программы и выход из GDB

## 4.5 Обработка аргументов командной строки в GDB

Скопировала файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm.

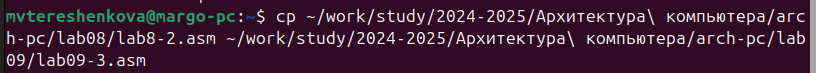


Рис. 19: Копирование файла

Создаю исполняемый файл.

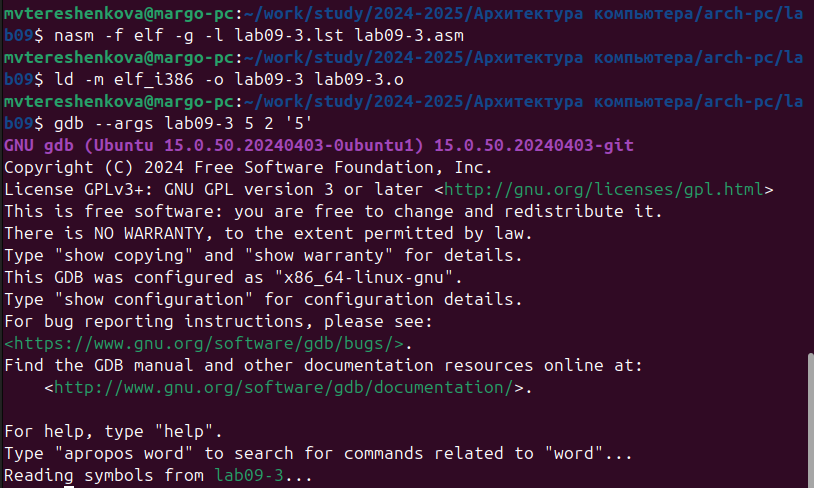


Рис. 20: Создание исполняемого файла

Для начала установила точку останова перед первой инструкцией в программе и запустим её.

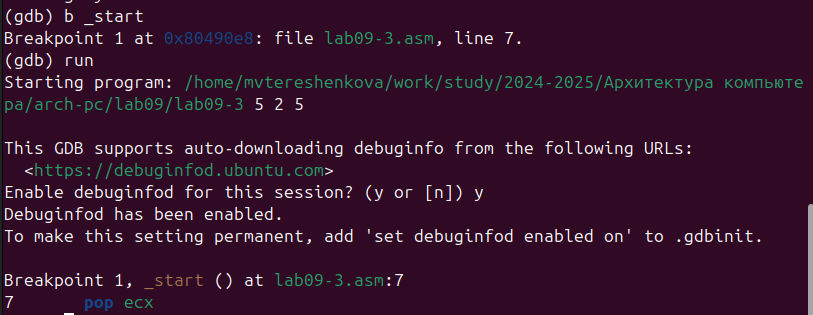


Рис. 21: Установление точки останова

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки.

Рис. 22: Количество аргументов командной строки

Рис. 22: Количество аргументов командной строки

Посмотрела остальные позиции стека – по адесу [esp+4] располагается адрес в памяти, где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по аресу [esp+12] – второго и т.д.

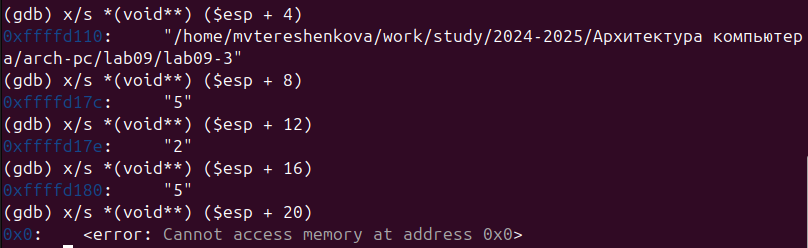


Рис. 23: Позиции стека

Шаг изменения адреса равен 4, потому что:

• Процессор обрабатывает данные 32-битными словами (4 байта).

• Аргументы на стеке выравниваются по границе 4 байт.

• Это упрощает доступ и обеспечивает корректность выполнения программы на архитектуре x86.

# 5 Задания для самостоятельной работы

1. Открываю программу из лабораторной работы №8 и начинаю её редактировать.

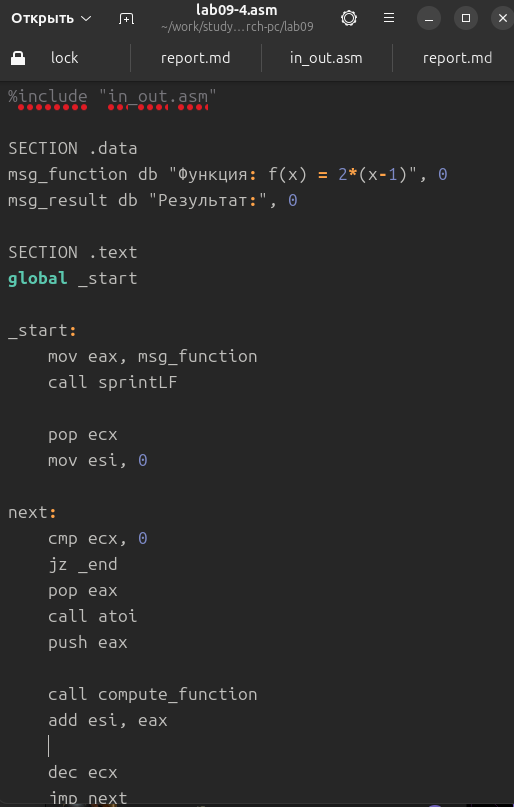


Рис. 24: Редактирование файла

Создаю исполняемый файл и запускаю его, чтобы проверить работу программы.

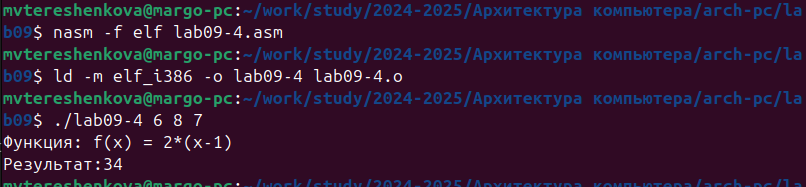


Рис. 25: Запуск исполняемого файла

Код программы:

%include “in\_out.asm”

SECTION .data

msg\_function db “Функция: f(x) = 2\*(x-1)“, 0 msg\_result db”Результат:“, 0

SECTION .text

global \_start

\_start:

mov eax, msg\_function

call sprintLF

pop ecx

pop edx

sub ecx, 1

mov esi, 0

next:

cmp ecx, 0

jz \_end

pop eax

call atoi

call \_calculate\_fx

add esi, eax

loop next

\_end:

mov eax, msg\_result

call sprint

mov eax, esi

call iprintLF

call quit

\_calculate\_fx:

sub eax, 1

mov ecx, 2

mul ecx

ret

1. Копирую код программы из листинга 9.3

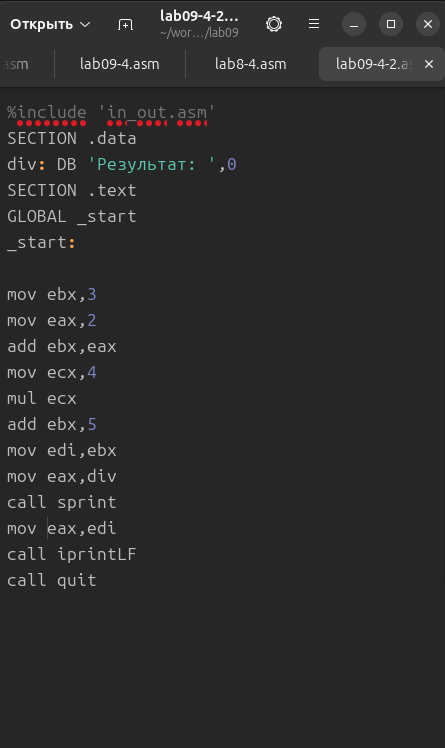


Рис. 26: Копирование файла из листинга 9.3

Запускаю программу в режике отладичка и пошагово через si просматриваю изменение значений регистров через i r.

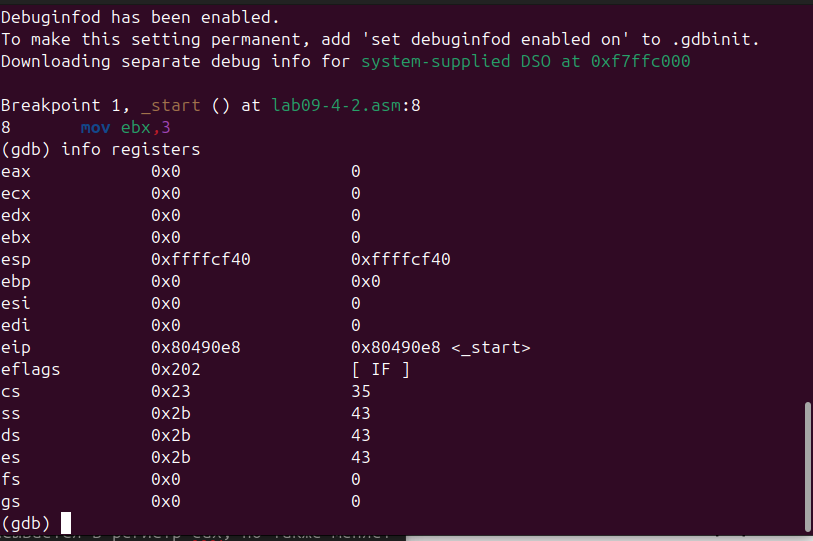


Рис. 27: Просмотр изменений регистров

При выполнении инструкции mul ecx можно заметить, что результат умножения записывается в регистр eax, но также меняет и edx. Значение регистра ebx не обновляется напрямую, поэтому результат программа неверно подсчитывает функцию.

Исправляю найденную ошибку, теперь программа верно считает значение функции.

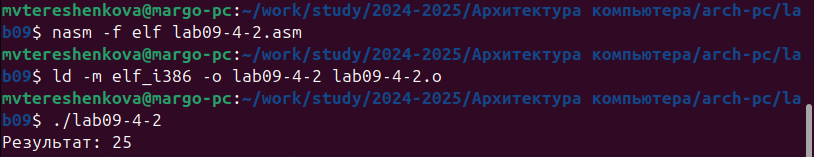


Рис. 28: Проверка работы программы

Код измененной программы:

%include ‘in\_out.asm’

SECTION .data

div: DB ‘Результат:’, 0

SECTION .text

GLOBAL \_start

\_start:

mov ebx, 3

mov eax, 2

add ebx, eax

mov eax, ebx

mov ecx, 4

mul ecx

add eax, 5

mov edi, eax

mov eax, div

call sprint

mov eax, edi

call iprintLF

call quit

# 6 Выводы

Благодаря данной лабораторной работе приобрела навыки написания программ с использованием подпрограмм; Познакомились с методами отладки при помощи GDB и его основными возможностями.

# Список литературы

1. Архитектура компьютеров