Отчёт по лабораторной работе №9

дисциплина: Архитектура компьютера

Аносов Даниил Игоревич

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. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятель- ной работы), реализовав вычисление значения функции как подпрограмму.
2. В листинге 9.3 приведена программа вычисления выражения . При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

# 3 Выполнение лабораторной работы

## 3.1 Реализация подпрограмм в NASM

Откроем терминал и создадим каталог для программ лабораторной работы №8. В новом каталоге создадим файл для первой программы *lab9-1.asm*. (рис. 1).

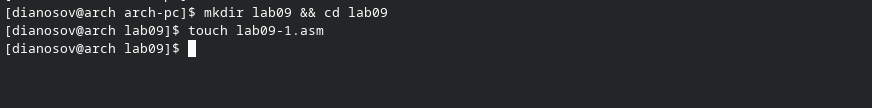


Рис. 1: Создание каталога для программ

Введём в этот файл текст программы из предложенного листинга. (рис. 2).

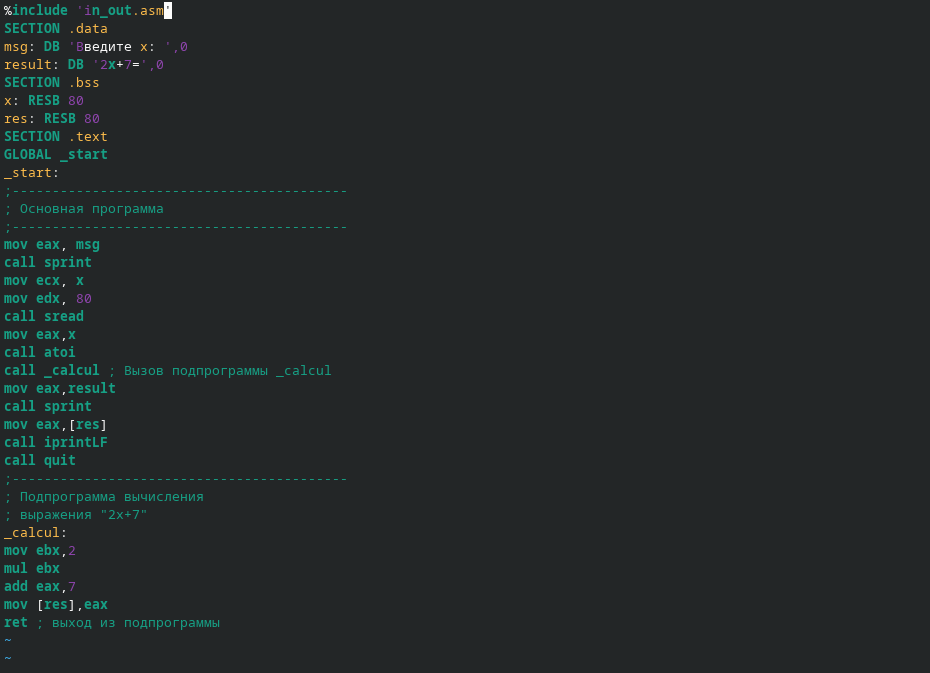


Рис. 2: Открытый Vim

Первые строки программы отвечают за вывод сообщения на экран (call sprint), чтение данных введенных с клавиатуры (call sread) и преобразования введенных данных из символьного вида в численный (call atoi).

Скомпилируем и запустим программу, предварительно скопировав из каталога предыдущей лабораторной работы вспомогательный файл с подпрограммами *in\_out.asm* (рис. 3).

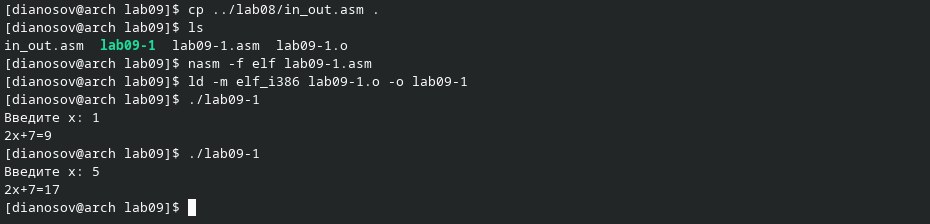


Рис. 3: Компиляция и первый запуск программы

Изменим текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения , где вводится с клавиатуры, , . Т.е. передается в подпрограмму \_calcul из нее в подпрограмму \_subcalcul, где вычисляется выражение , результат возвращается в \_calcul и вычисляется выражение . Результат возвращается в основную программу для вывода результата на экран. (рис. 4).

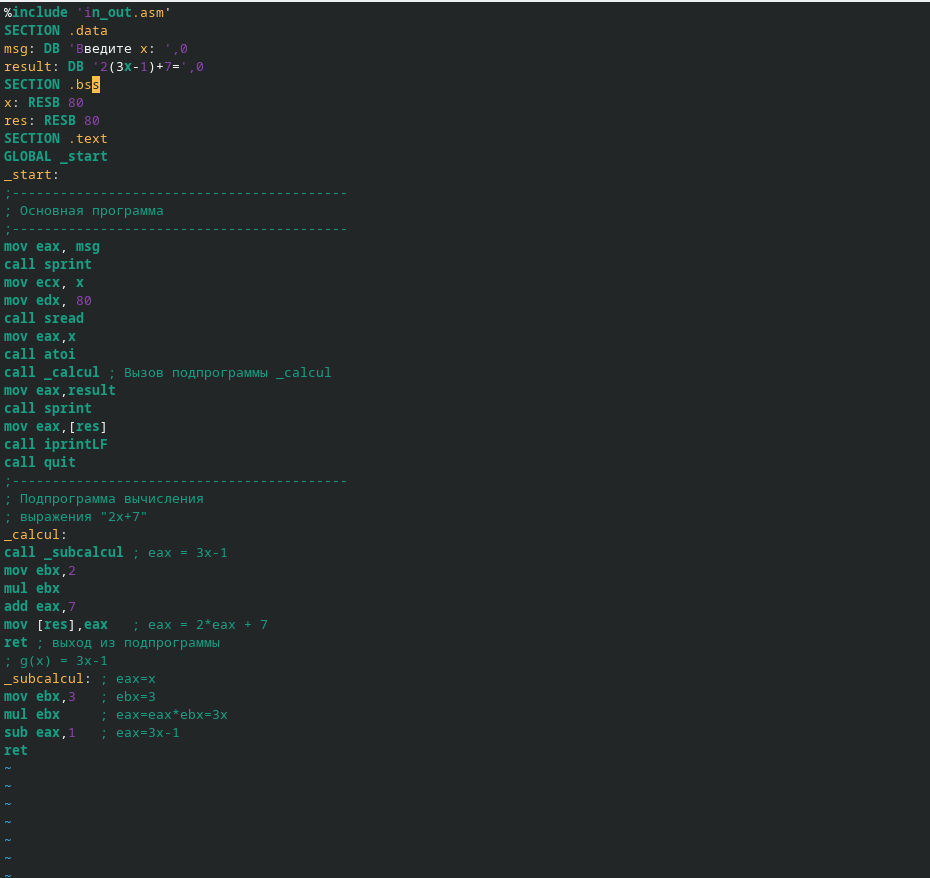


Рис. 4: Vim с обновленной программой

Скомпилируем и запустим измененную программу. Проверим её работу. (рис. 5).

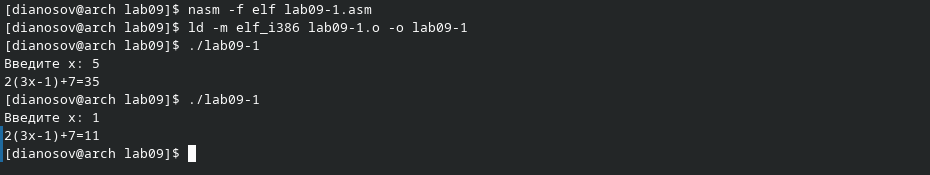


Рис. 5: Повторная компиляция и запуск программы

Теперь, после того, как мы добавили новую подпрограмму \_subcalcul, программа корректно выполняет свою задачу.

Создадим новый файл *lab09-2.asm*. Введём в него код из предложенного листинга.

Откроем файл программы в **Vim** и отредактируем файл (рис. 6).

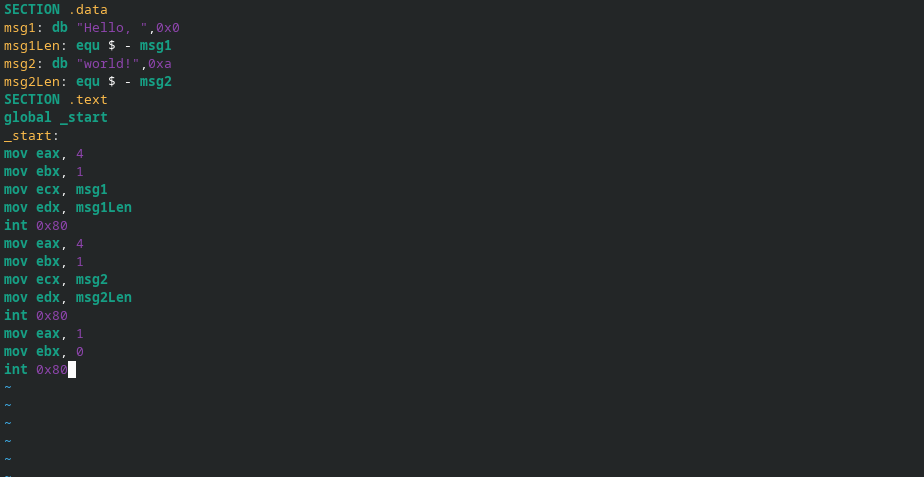


Рис. 6: Vim с новой программой

Проведём компиляцию новой программы, получим исполняемый файл (рис. 7).

## 3.2 Отладка программам с помощью GDB

Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’:

nasm -f elf -g -l lab09-2.lst lab09-2.asm  
ld -m elf\_i386 -o lab09-2 lab09-2.o

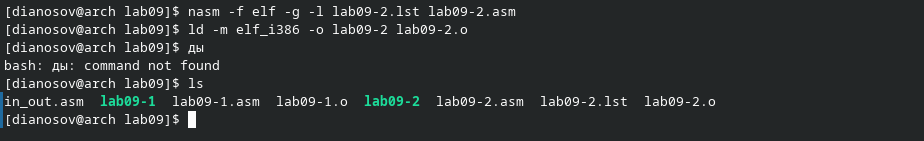


Рис. 7: Компиляция и запуск новой программы

Откроем отладчик **GDB** (рис. 8). В нём исследуем, как работают брейкпоинты (точки останова).

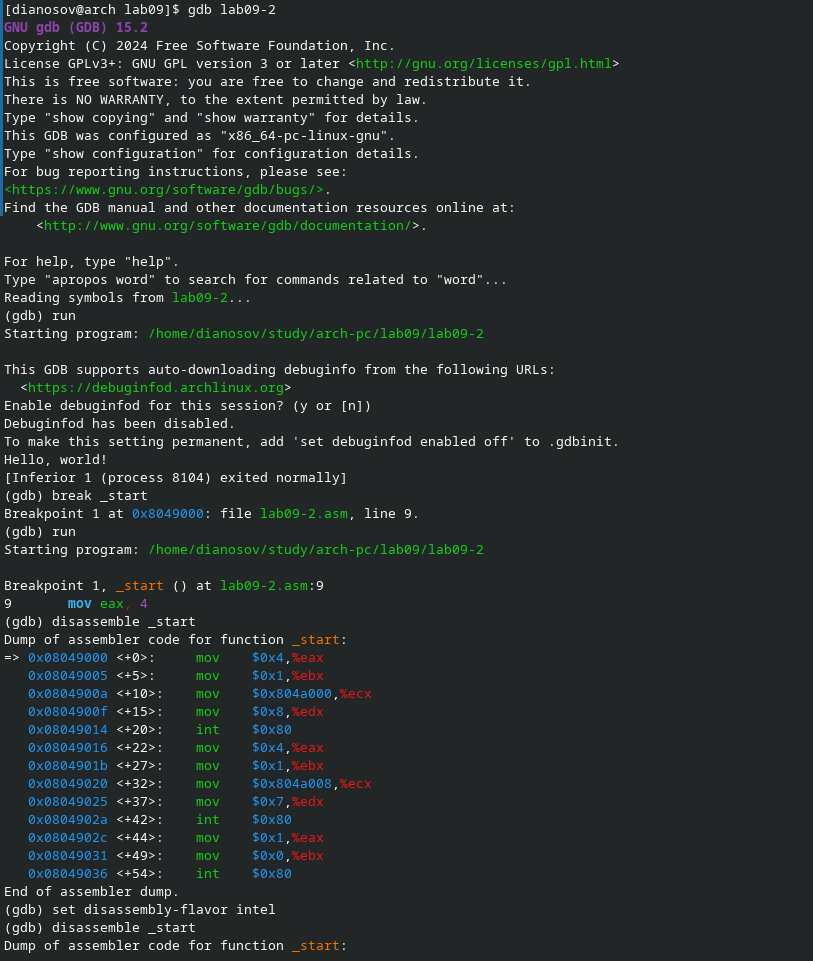


Рис. 8: Открытый **GDB**

Переключимся на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. 9). Этот режим отличается от режима *ATT* порядком операндов и стилем их обозначений, а именно, в *ATT* перед именами регистров стоят $. Порядок операндов: в *ATT* “source, destination”, а в *Intel* “destination, source”.

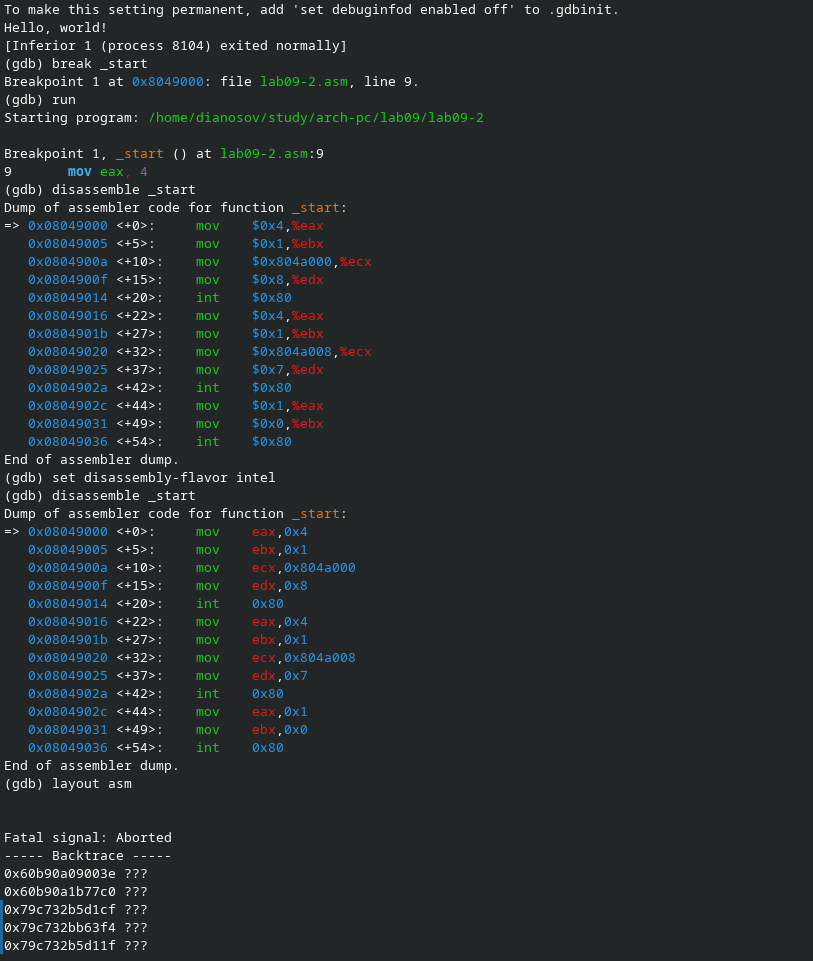


Рис. 9: Открытый **GDB**

Теперь откроем режим псевдографики для более удобного анализа программы (рис. 10).

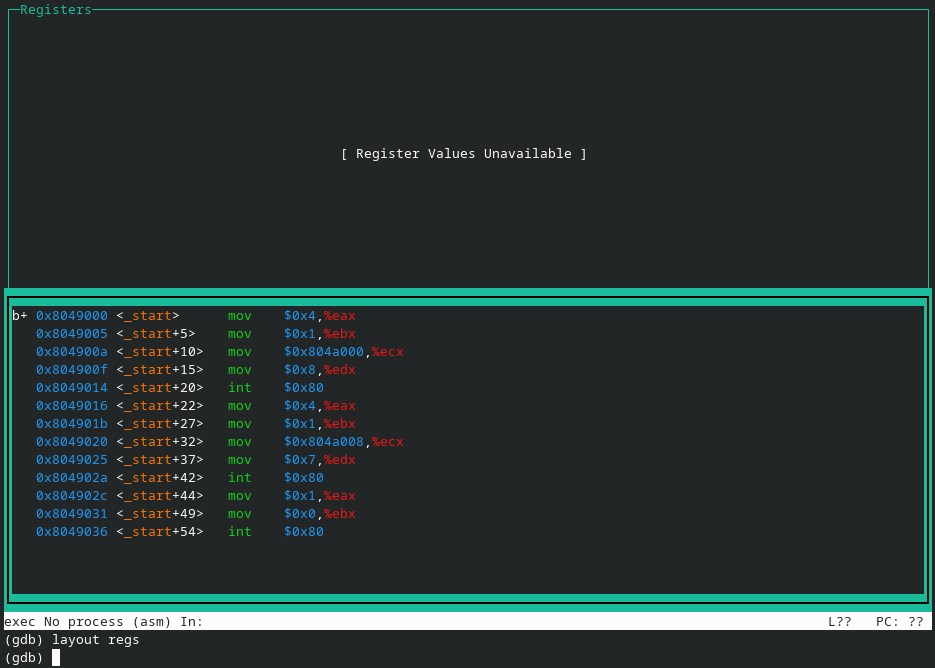


Рис. 10: Режим псевдографики

### 3.2.1 Добавление точек останова

Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»: На предыдущих шагах была установлена точка останова по имени метки (\_start). Про- верим это с помощью команды info breakpoints (кратко i b). Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции (см. рис. 9.3). Определим адрес предпоследней инструкции (mov ebx,0x0) и установим точку останова (рис. 11).



Рис. 11: Работа в **GDB**

### 3.2.2 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполним 5 инструкций с помощью команды stepi (или si) и проследим за изменением значений регистров. Значения каких регистров изменяются? Посмотреть содержимое регистров также можно с помощью команды info registers (или i r).

(gdb) info registers

Для отображения содержимого памяти можно использовать команду x <адрес>, которая выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором выводятся данные, можно задать после имени команды через косую черту: x/NFU <адрес>. С помощью команды x &<имя переменной> также можно посмотреть содержимое переменной. Посмотрите значение переменной msg1 по имени

(gdb) x/1sb &msg1  
0x804a000 <msg1>: "Hello, "

Посмотрите значение переменной msg2 по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрим инструкцию mov ecx,msg2 которая записывает в регистр ecx адрес перемененной msg2. (рис. 12, 13).

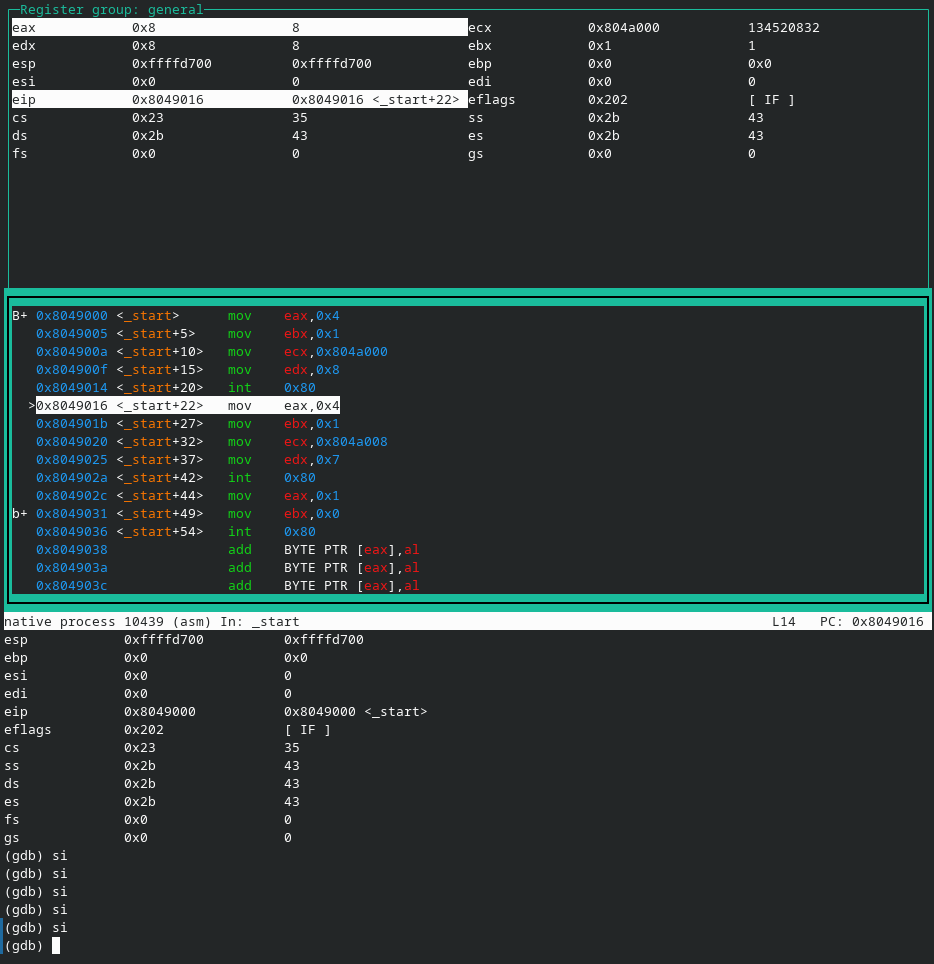


Рис. 12: Работа в **GDB**

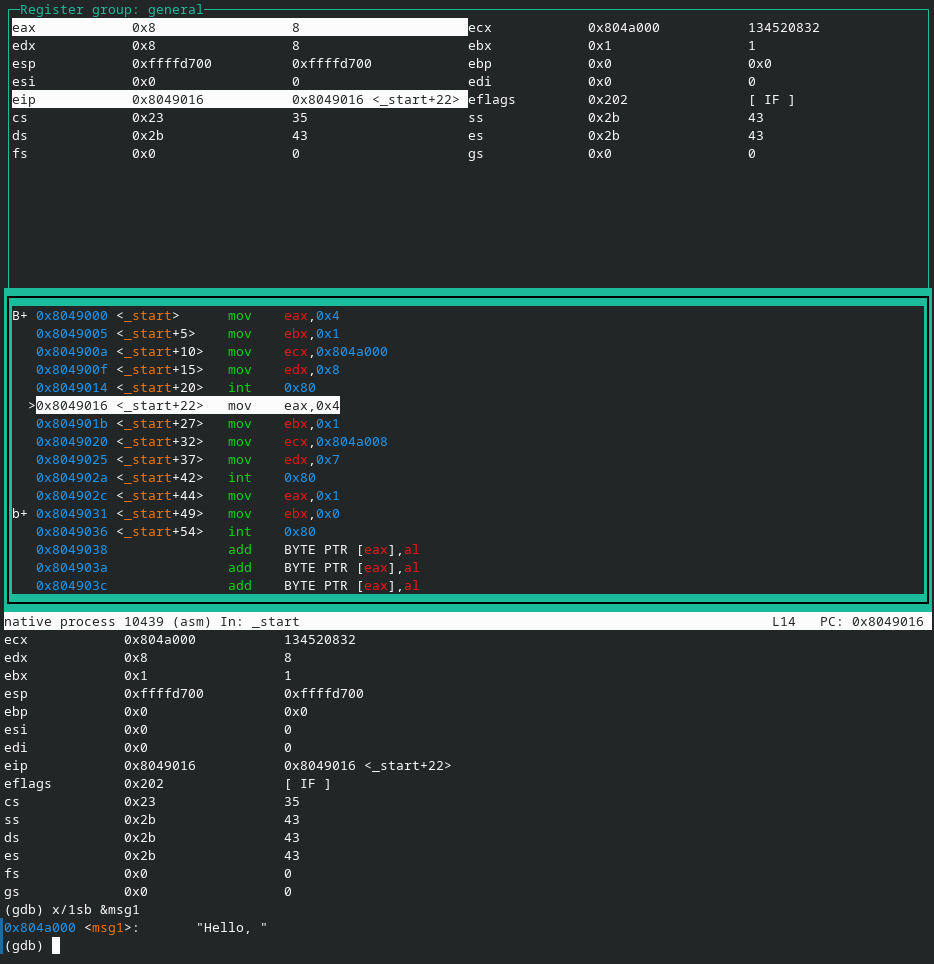


Рис. 13: Работа в **GDB**

### 3.2.3 Обработка аргументов командной строки в GDB

Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем lab09-3.asm, создадим исполняемый файл (рис. 14).

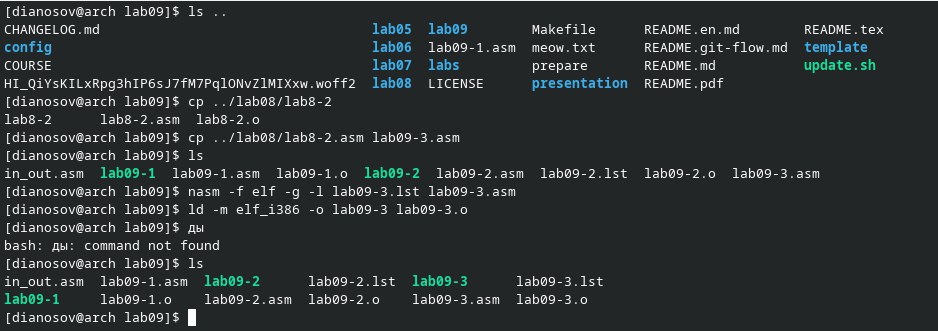


Рис. 14: Открытый терминал

Для загрузки в gdb программы с аргументами необходимо использовать ключ --args. Загрузим исполняемый файл в отладчик, указав аргументы (рис. 15):

gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'

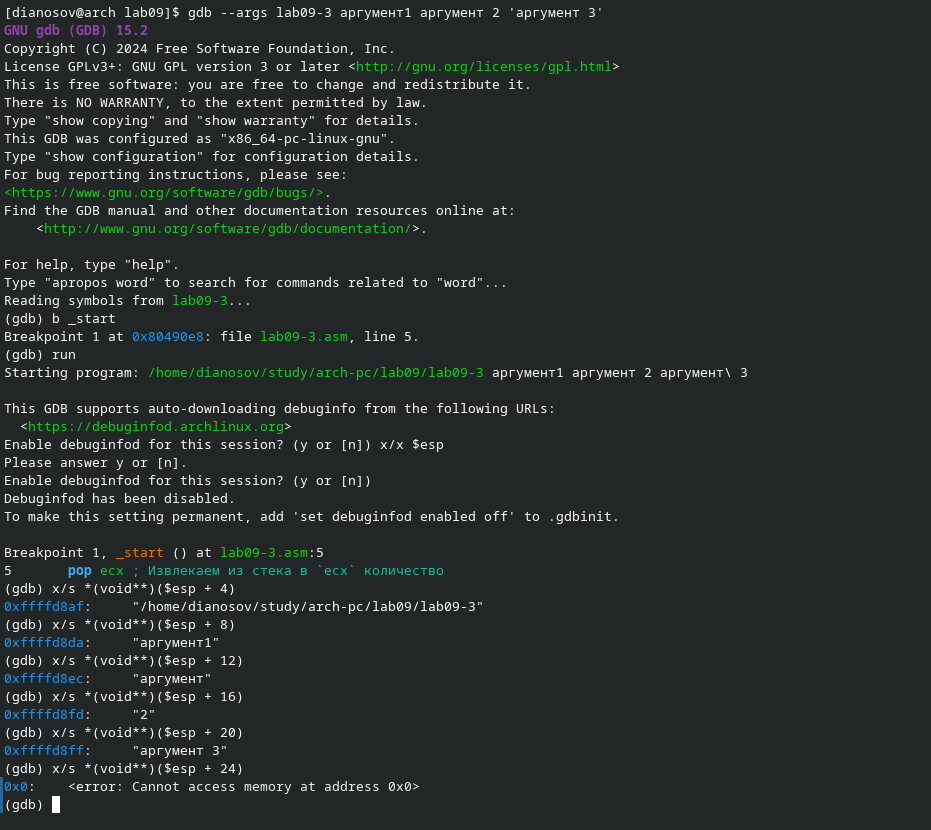


Рис. 15: Открытый **GDB**

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Посмотрим остальные позиции стека – по адесу [esp+4] располагается адрес в памяти где находится имя программы, по адесу [esp+8] хранится адрес первого аргумента, по адресу [esp+12] – второго и т.д.

Аргументы расположены в памяти с шагом 4 потому что в программе под каждый из них выделено по 4 байта.

# 4 Задание для самостоятельной работы

## 4.1 Первое задание

Скопируем из каталога последней лабораторной работы файл *task.asm*.

Откроем его в редакторе **Vim** (рис. 16).

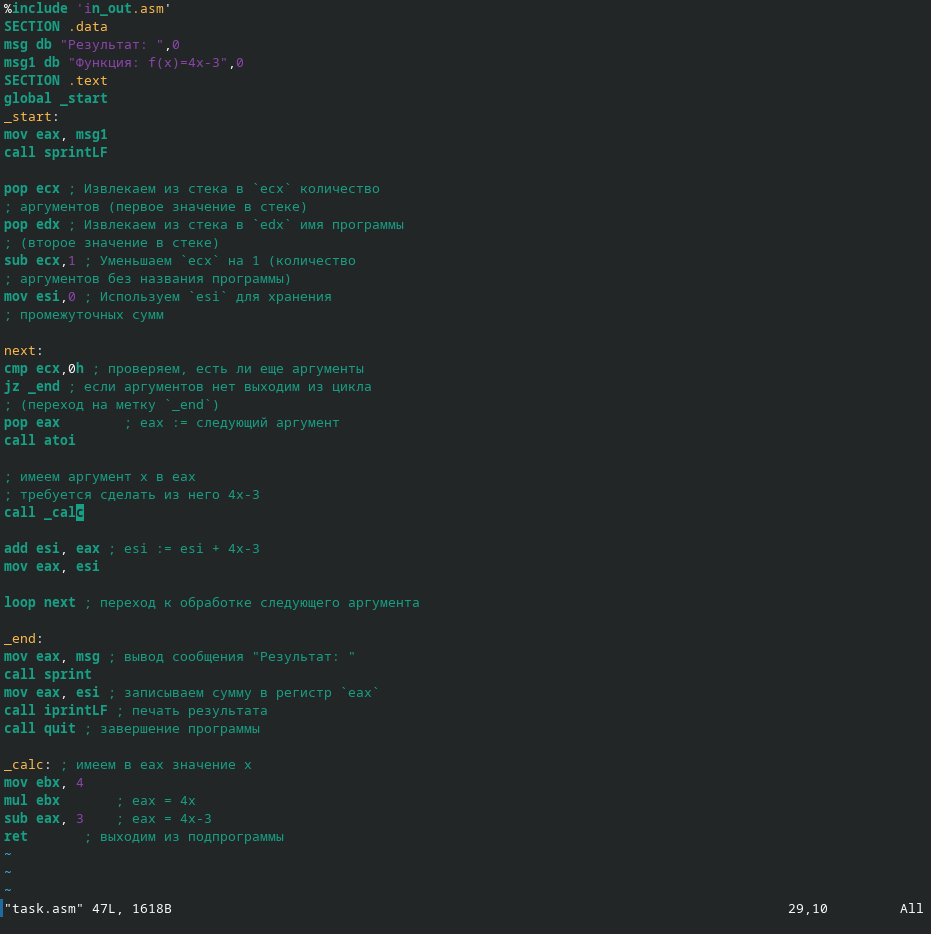


Рис. 16: **Vim** с файлом *task.asm*

Реализуем с помощью подпрограммы вычисление значения выражения . Для этого добавим подпрограмму \_calc, где, имея в регистре eax значение переменной , преобразуем его. Затем инструкцией ret выйдем из подпрограммы, оставив результат в eax.

Протестируем работу программы (рис. 17).

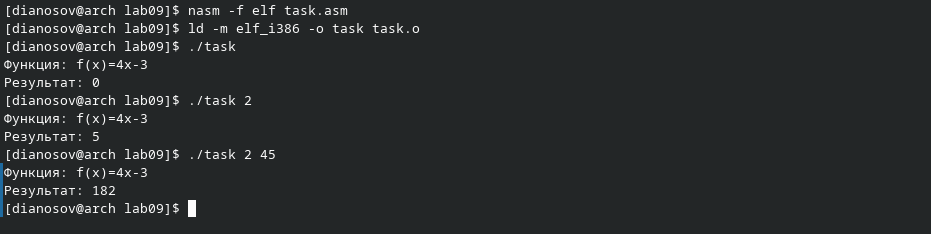


Рис. 17: Проверка работы *task.asm*

Выводится верный результат.

## 4.2 Второе задание

В листинге предложен код программы, в котором надо найти и исправить ошибку.

%include 'in\_out.asm'  
SECTION .data  
div: DB 'Результат: ',0  
SECTION .text  
GLOBAL \_start  
\_start:  
; ---- Вычисление выражения (3+2)\*4+5  
mov ebx,3  
mov eax,2  
add ebx,eax  
mov ecx,4  
mul ecx  
add ebx,5  
mov edi,ebx  
; ---- Вывод результата на экран  
mov eax,div  
call sprint  
mov eax,edi  
call iprintLF  
call quit

Будем использовать для этого отладчик **GDB** (рис. 18). Установим точку останова на метке \_start, войдём в режим отображения значений регистров, поменяем стиль имён регистров на intel.

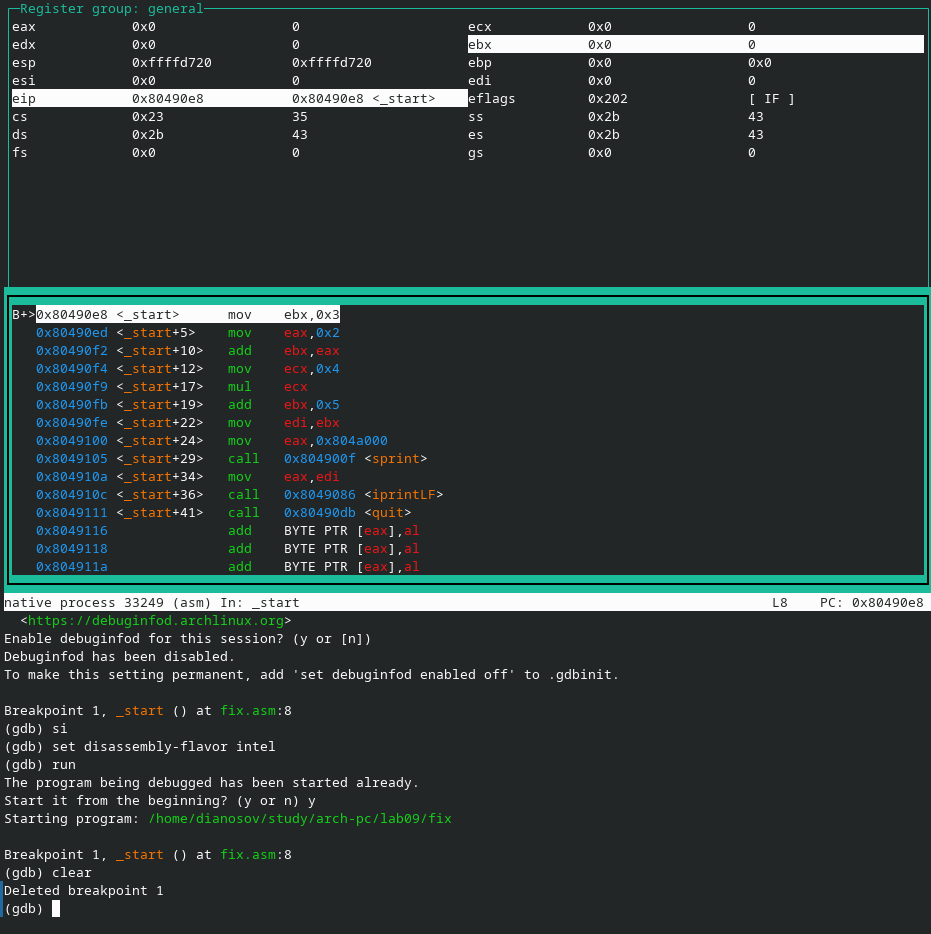


Рис. 18: **GDB** с *fix.asm*

Напомним: программа должна вычислять значение выражения Пройдём последовательно на несколько инструкций вперёд повторным вводом команды si (рис. 19).

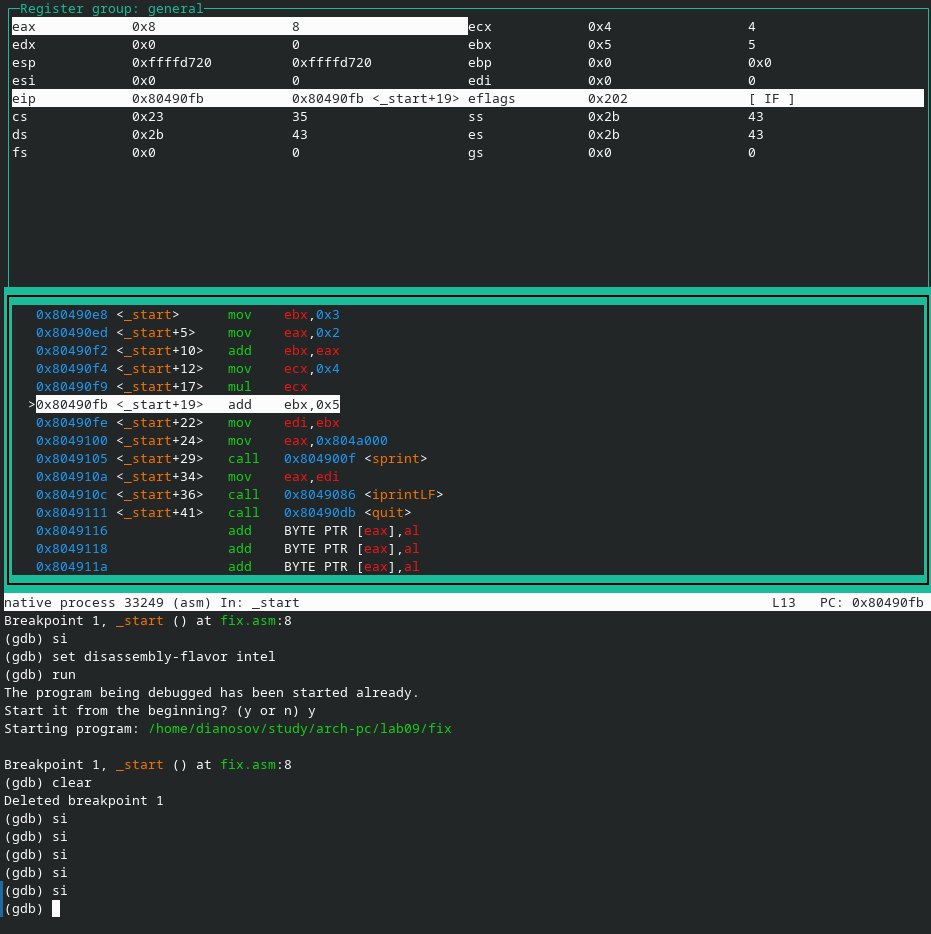


Рис. 19: **GDB** с *fix.asm*

Видим, что на умножается не сумма , лежащая в регистре ebx, а только число в регистре eax. Это происходит из-за того, что инструкция mul x умножает значение регистра eax на значение x. Чтобы исправить ошибку, поменяем инструкцию add ebx, eax на add eax, ebx, чтобы значение помещалось не в ebx, а не в eax. Также, поскольку теперь мы имеем результат в регистре eax, поменяем add ebx, 5 на add eax, 5 и mov edi, ebx на mov edi, eax.

Изменив код, проверим корректность его работы (рис. 20).

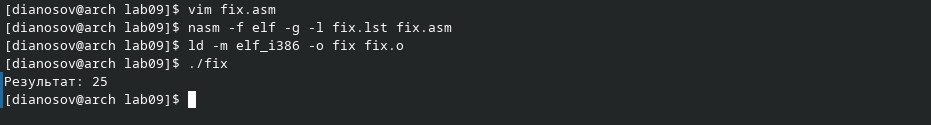


Рис. 20: Тестирование исправленной программы *fix.asm*

Программа работает корректно. Задание выполнено.

Загрузим файлы на GitHub (рис. 21).

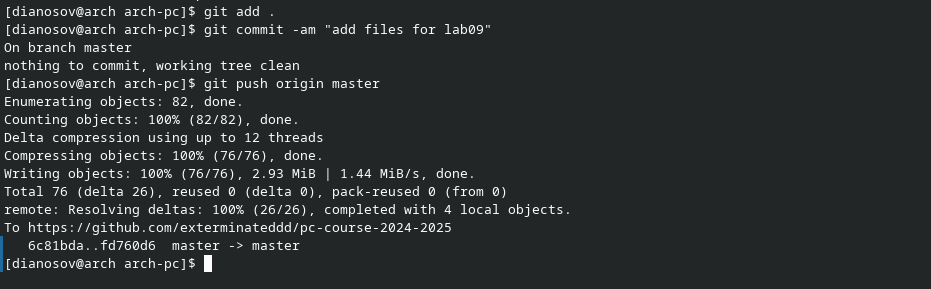


Рис. 21: Загрузка файлов на GitHub

# 5 Выводы

Приобретены навыки написания программ с использованием подпрограмм. Освоены методы отладки при помощи GDB и его основные возможности.