Лабораторная работа №9

Архитектура компьютера

Мурашов Иван Вячеславович

Содержание

# 1 Цель работы

Целью данной лабораторной работы является приобретение навыков написания программ с использованием подпрограмм, знакомство с методами отладки при помощи GDB и его основными возможностями.

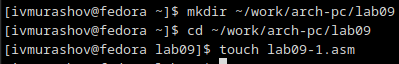
# 2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программам с помощью GDB
3. Добавление точек останова
4. Работа с данными программы в GDB
5. Обработка аргументов командной строки в GDB
6. Выполнение заданий для самостоятельной работы

# 3 Выполнение лабораторной работы

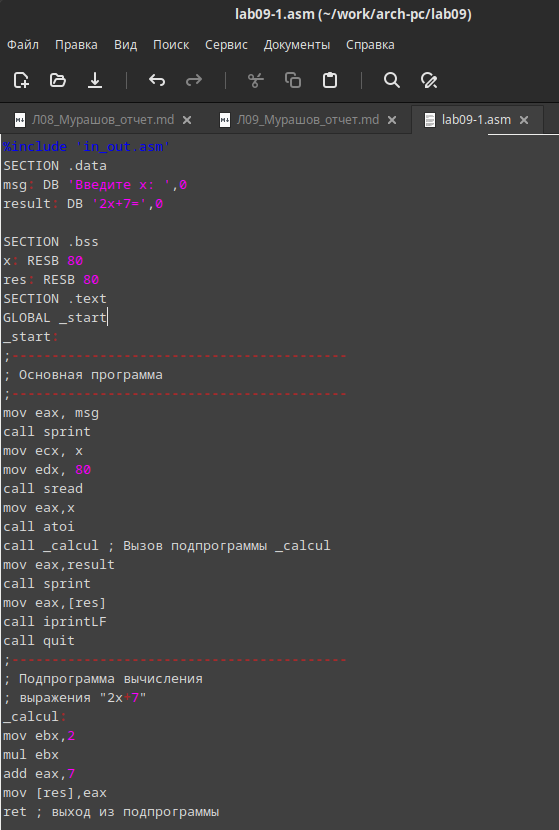
## 3.1 Реализация подпрограмм в NASM

Создаю каталог для программам лабораторной работы №9, перехожу в него и создаю файл lab09-1.asm (рис. [??]).



Создание каталога и файла в нём

Ввожу в файл lab09-1.asm текст программы из листинга 9.1 (рис. [??]).

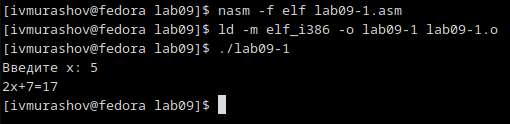


Редактирование файла

**Листинг 1. Пример программы с использованием вызова подпрограммы**

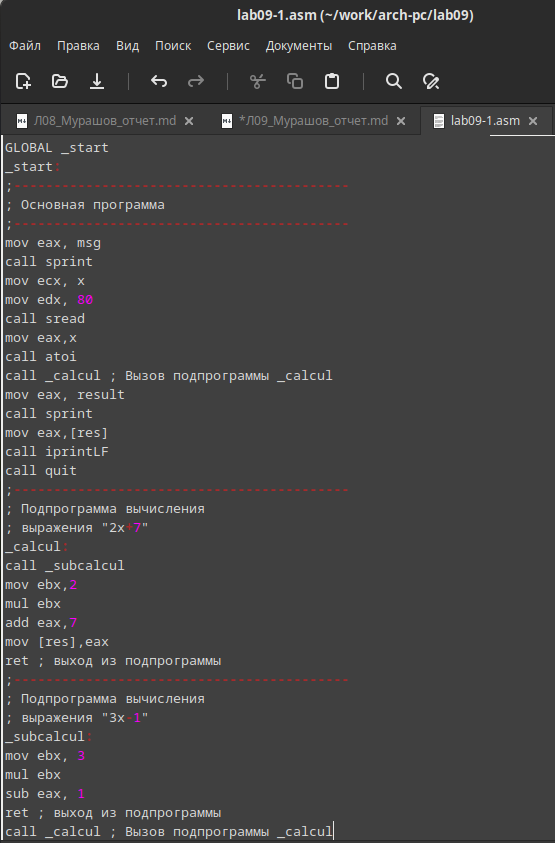
%include 'in\_out.asm'  
SECTION .data  
msg: DB 'Введите x: ',0  
result: DB '2x+7=',0  
  
SECTION .bss  
x: RESB 80  
res: RESB 80  
SECTION .text  
GLOBAL \_start  
\_start:  
;------------------------------------------  
; Основная программа  
;------------------------------------------  
mov eax, msg  
call sprint  
mov ecx, x  
mov edx, 80  
call sread  
mov eax,x  
call atoi  
call \_calcul ; Вызов подпрограммы \_calcul  
mov eax,result  
call sprint  
mov eax,[res]  
call iprintLF  
call quit  
;------------------------------------------  
; Подпрограмма вычисления  
; выражения "2x+7"  
\_calcul:  
mov ebx,2  
mul ebx  
add eax,7  
mov [res],eax  
ret ; выход из подпрограммы

Создаю исполняемый файл и запускаю его. Проверяю работу программы, вручную посчитав искомое значение. Результаты совпали (рис. [??]).



Трансляция, компоновка и запуск файлов

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. Т.е. x передается в подпрограмму \_calcul из нее в подпрограмму \_subcalcul, где вычисляется выражение g(x), результат возвращается в \_calcul и вычисляется выражение f(g(x)). Результат возвращается в основную программу для вывода результата на экран (рис. [??]).

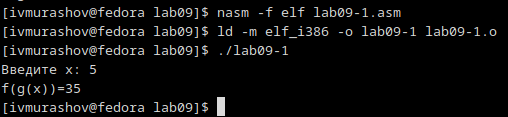


Редактирование файла

**Листинг 1.1. Программа с использованием вызова подпрограммы и её подпрограммы**

%include 'in\_out.asm'  
SECTION .data  
msg: DB 'Введите x: ', 0  
result: DB 'f(g(x))=', 0  
  
SECTION .bss  
x: RESB 80  
res: RESB 80  
SECTION .text  
GLOBAL \_start  
\_start:  
;------------------------------------------  
; Основная программа  
;------------------------------------------  
mov eax, msg  
call sprint  
mov ecx, x  
mov edx, 80  
call sread  
mov eax,x  
call atoi  
call \_calcul ; Вызов подпрограммы \_calcul  
mov eax, result  
call sprint  
mov eax,[res]  
call iprintLF  
call quit  
;------------------------------------------  
; Подпрограмма вычисления  
; выражения "2x+7"  
\_calcul:  
call \_subcalcul  
mov ebx,2  
mul ebx  
add eax,7  
mov [res],eax  
ret ; выход из подпрограммы  
;------------------------------------------  
; Подпрограмма вычисления  
; выражения "3x-1"  
\_subcalcul:  
mov ebx, 3  
mul ebx  
sub eax, 1  
ret ; выход из подпрограммы  
call \_calcul ; Вызов подпрограммы \_calcul

Создаю исполняемый файл и запускаю его. Проверяю работу программы, вручную посчитав искомое значение. Результаты совпали(рис. [??]).



Трансляция, компоновка и запуск файлов

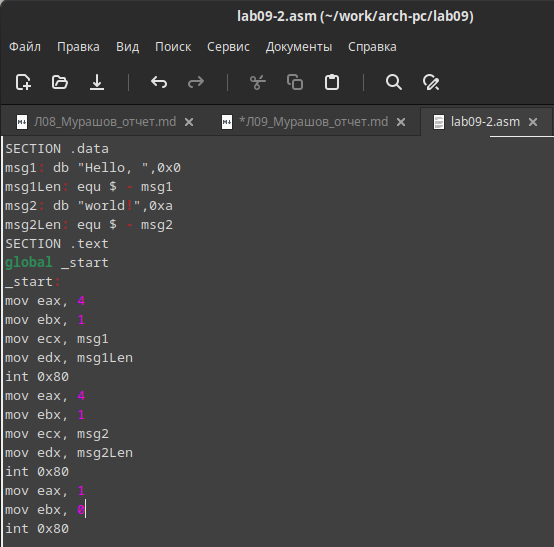
## 3.2 Отладка программам с помощью GDB

Создаю файл lab09-2.asm в каталоге ~/work/arch-pc/lab09 (рис. [??]).

Создание файла

Создание файла

Ввожу в файл lab09-2.asm текст программы из листинга 9.2 (рис. [??]).



Редактирование файла

**Листинг 2. Программа вывода сообщения Hello world!**

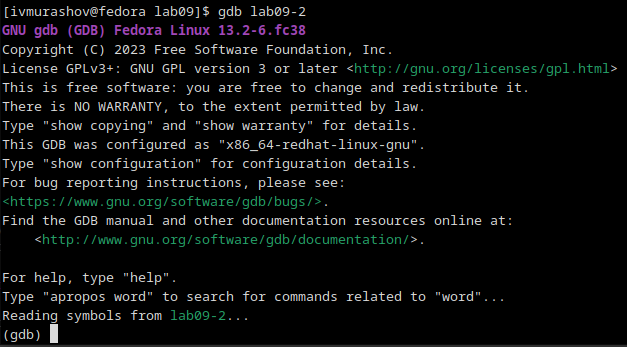
SECTION .data  
msg1: db "Hello, ",0x0  
msg1Len: equ $ - msg1  
msg2: db "world!",0xa  
msg2Len: equ $ - msg2  
SECTION .text  
global \_start  
\_start:  
mov eax, 4  
mov ebx, 1  
mov ecx, msg1  
mov edx, msg1Len  
int 0x80  
mov eax, 4  
mov ebx, 1  
mov ecx, msg2  
mov edx, msg2Len  
int 0x80  
mov eax, 1  
mov ebx, 0  
int 0x80

Получаю исполняемый файл. Для работы с GDB добавляю в исполняемый файл отладочную информацию, для этого провожу трансляцию программ с ключом ‘-g’ (рис. [??]).

Трансляция, компоновка и запуск файлов

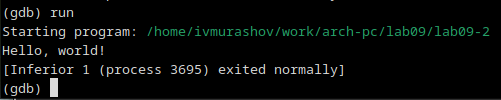
Трансляция, компоновка и запуск файлов

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



Загрузка исполняемого файла в отладчик gdb

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



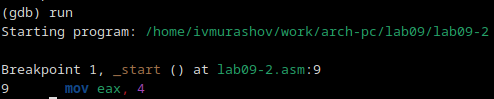
Запуск программы в оболочке GDB

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы (рис. [??]).

Установка брейкпоинта в оболочке GDB

Установка брейкпоинта в оболочке GDB

Запускаю программу (рис. [??]).



Запуск программы в оболочке GDB

Просматриваю дизассимилированный код программы с помощью команды disassemble, начиная с метки \_start (рис. [??]).



Просмотр дизассимилированного кода программы в оболочке GDB

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. [??]).

Переключение синтаксиса при отображении команд в оболочке GDB

Переключение синтаксиса при отображении команд в оболочке GDB

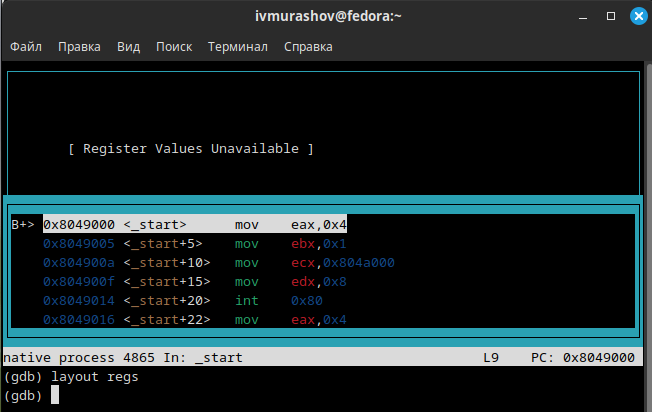
Просматриваю дизассимилированный код программы, начиная с метки \_start (рис. [??]).



Просмотр дизассимилированного кода программы в оболочке GDB

Cинтаксис машинных команд в режимe ATT включает в себя использование значка ‘$’ перед операндами, значка ‘%’ перед регистрами в то время как синтаксис в режиме Intel этих значков нет. Также в режиме ATT сначала указываются операнды, а потом регистры. В Intel наоборот - сначала регистры, затем операнды.

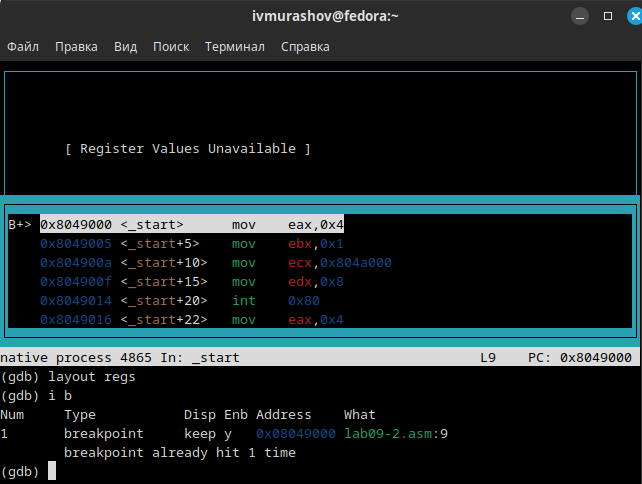
Включаю режим псевдографики для более удобного анализа программы рис. [??]).



Режим псевдографики GDB

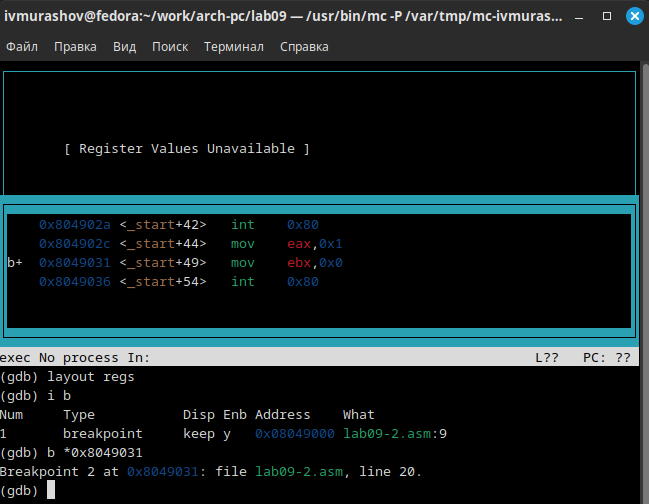
## 3.3 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяю это с помощью команды info breakpoints (кратко i b) (рис. [??]).



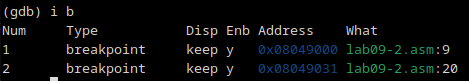
Получение информации о точках останова в оболочке GDB

Определяю адрес предпоследней инструкции (mov ebx, 0x0) и устанавливаю точку останова рис. [??]).



Установка точки останова по адресу инструкции в оболочке GDB

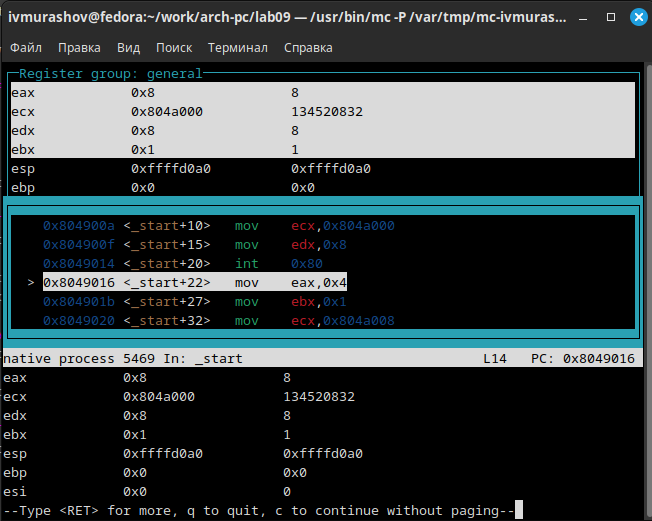
Просматриваю информацию о всех установленных точках останова (рис. [??]).



Получение информации о точках останова в оболочке GDB

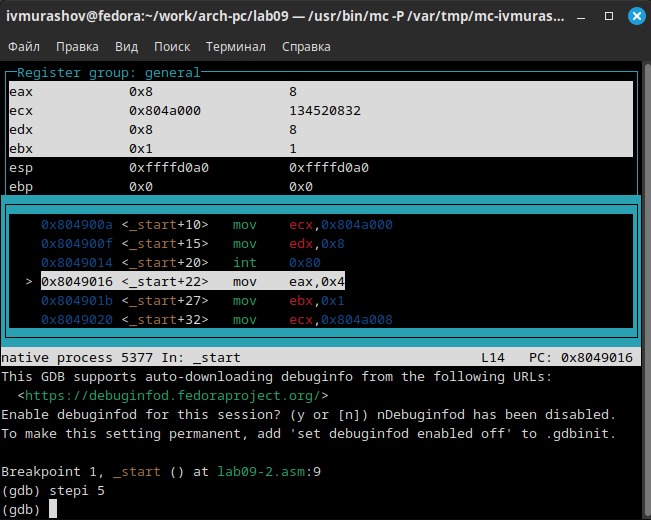
## 3.4 Работа с данными программы в GDB

Просматриваю содержимое регистров с помощью команды info registers (или i r) (рис. [??]).



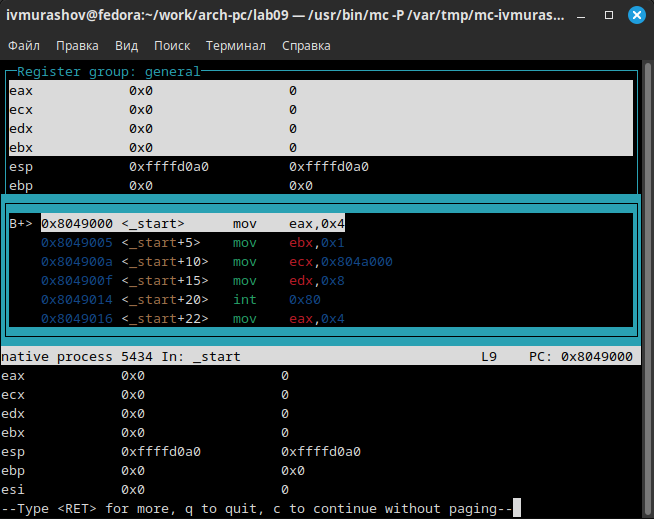
Получение информации о точках останова в оболочке GDB

Выполняю 5 инструкций с помощью команды stepi (или si) (рис. [??]).



Выполнение 5 шагов программы в оболочке GDB

Просматриваю содержимое регистров с помощью команды info registers (или i r) (рис. [??]).



Получение информации о точках останова в оболочке GDB

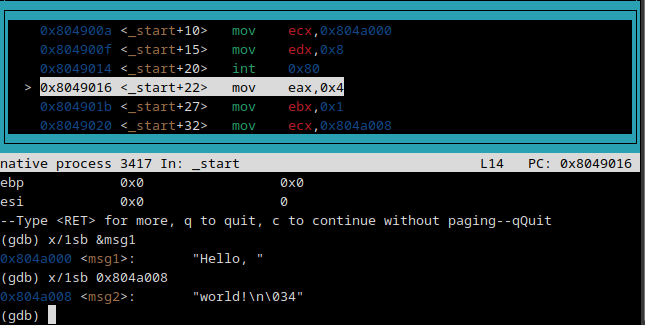
Изменились значения регистров eax, ecx, edx, ebx.

Просматриваю значение переменной msg1 по имени (рис. [??]).

Работа с переменными в оболочке GDB

Работа с переменными в оболочке GDB

Просматриваю значение переменной msg2 по адресу, определяя его по дизассемблированной инструкции (рис. [??]).



Работа с переменными в оболочке GDB

Изменяю первый символ переменной msg1 и просматриваю значение msg1 (рис. [??]).

Работа с переменными в оболочке GDB

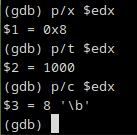
Работа с переменными в оболочке GDB

Заменяю 3 символ в переменной msg2 и просматриваю значение msg2 (рис. [??]).

Работа с переменными в оболочке GDB

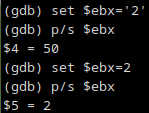
Работа с переменными в оболочке GDB

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



Работа с регистрами в оболочке GDB

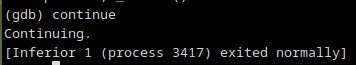
С помощью команды set изменяю значение регистра ebx (рис. [??]).



Работа с регистрами в оболочке GDB

При изменении значения регистра ebx на ‘2’ мы переводим символ в строковый вид (по таблице ASCII символ ‘2’ соответствует 50 а десятичном представлении), а при изменении на 2 число остаётся в строковом виде.

Завершаю выполнение программы с помощью команды continue (сокращенно c) и выхожу из GDB с помощью команды quit (сокращенно q) (рис. [??]).



Работа в оболочке GDB

## 3.5 Обработка аргументов командной строки в GDB

Копирую 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. [??]).

Копирование файла

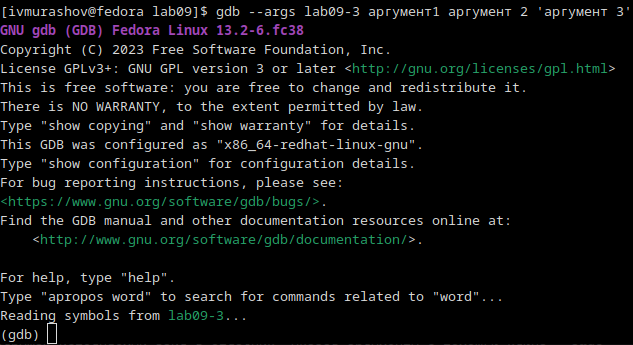
Копирование файла

Создаю исполняемый файл (рис. [??]).

Трансляция и компоновка файлов

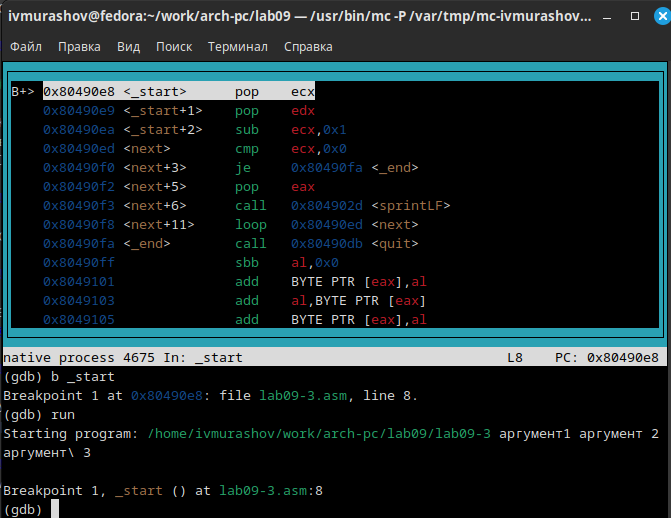
Трансляция и компоновка файлов

Загружаю исполняемый файл в отладчик, указав аргументы с помощью ключа –args (рис. [??]).



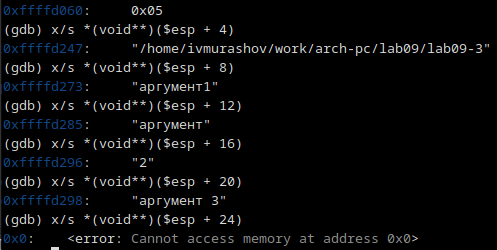
Загрузка исполняемого файла в отладчик gdb

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



Установка точки останова и запуск программы в оболочке GDB

Просматриваю позиции стека (рис. [??]).



Просмотр позиций стека в оболочке GDB

Шаг изменения адреса равен 4, так как

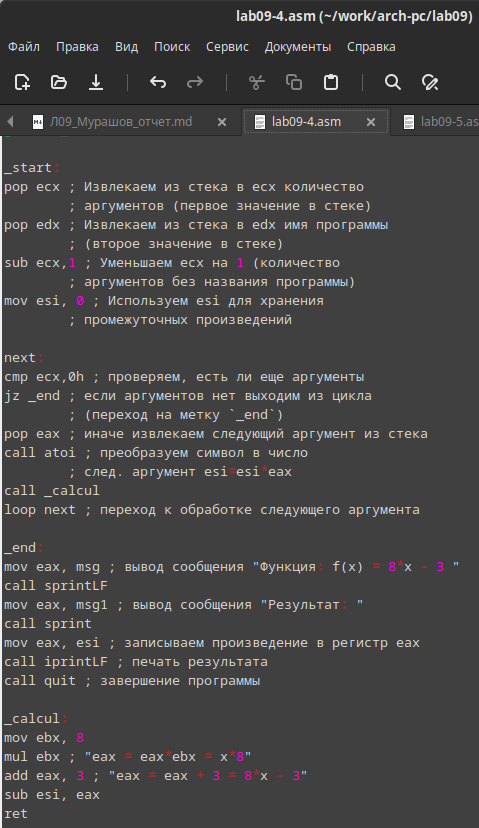
## 3.6 Выполнение заданий для самостоятельной работы

1. Копирую файл lab8-5.asm в каталог lab09 с новым именем lab09-4.asm (рис. [??]).

Копирование файла

Копирование файла

Изменяю программу, реализовав вычисление значения функции f(x) как подпрограмму (рис. [??]).

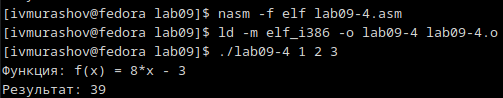


Редактирование файла

**Листинг 3. Программа вычисления суммы функций для аргументов командной строки**

%include 'in\_out.asm'   
SECTION .data   
msg db "Функция: f(x) = 8\*x - 3",0   
msg1 db "Результат: ",0   
SECTION .text   
global \_start   
   
\_start:   
pop ecx ; Извлекаем из стека в ecx количество   
 ; аргументов (первое значение в стеке)   
pop edx ; Извлекаем из стека в edx имя программы   
 ; (второе значение в стеке)   
sub ecx,1 ; Уменьшаем ecx на 1 (количество   
 ; аргументов без названия программы)   
mov esi, 0 ; Используем esi для хранения   
 ; промежуточных произведений   
   
next:   
cmp ecx,0h ; проверяем, есть ли еще аргументы   
jz \_end ; если аргументов нет выходим из цикла   
 ; (переход на метку `\_end`)   
pop eax ; иначе извлекаем следующий аргумент из стека   
call atoi ; преобразуем символ в число   
 ; след. аргумент esi=esi\*eax   
call \_calcul   
loop next ; переход к обработке следующего аргумента   
   
\_end:   
mov eax, msg ; вывод сообщения "Функция: f(x) = 8\*x - 3 "   
call sprintLF   
mov eax, msg1 ; вывод сообщения "Результат: "   
call sprint   
mov eax, esi ; записываем произведение в регистр eax   
call iprintLF ; печать результата   
call quit ; завершение программы   
   
\_calcul:   
mov ebx, 8   
mul ebx ; "eax = eax\*ebx = x\*8"   
sub eax, 3 ; "eax = eax - 3 = 8\*x - 3"   
add esi, eax  
ret

Проверяю корректность работы программы (рис. [??]).



Трансляция, компоновка и запуск файлов

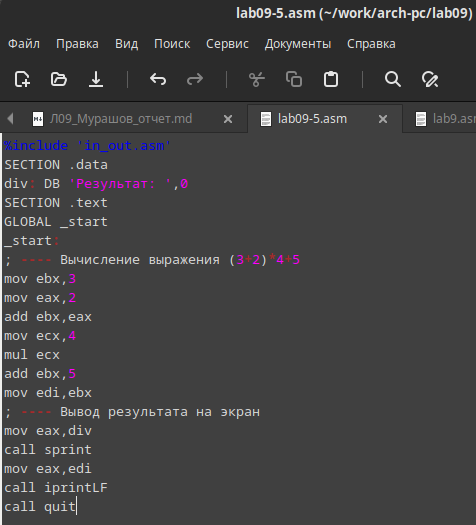
Программа работает корректно.

1. Создаю файл lab09-5.asm в каталоге lab09 (рис. [??]).

Создание файла

Создание файла

Ввожу в данный файл программу вычисления выражения (3+2)\*4+5 из листинга 9.3 (рис. [??]).



Редактирование файла

\*\*Листинг 4. Программа вычисления выражения (3+2)\*4+5)\*\*

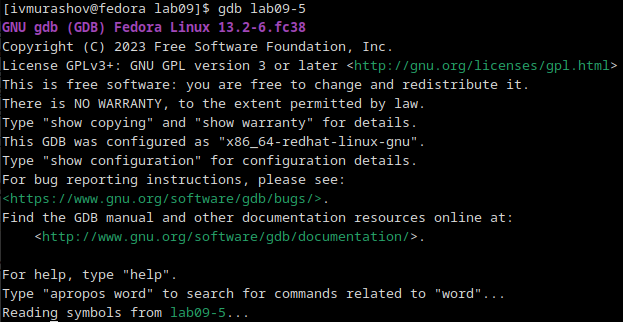
%include 'in\_out.asm'  
SECTION .data  
div: DB 'Результат: ',0  
SECTION .text  
GLOBAL \_start  
\_start:  
; ---- Вычисление выражения (3+2)\*4+5  
mov ebx,3  
mov eax,2  
add ebx,eax  
mov ecx,4  
mul ecx  
add ebx,5  
mov edi,ebx  
; ---- Вывод результата на экран  
mov eax,div  
call sprint  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

Получаю исполняемый файл (рис. [??]).

Трансляция, компоновка и запуск файлов

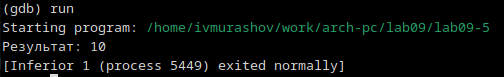
Трансляция, компоновка и запуск файлов

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



Загрузка исполняемого файла в отладчик gdb

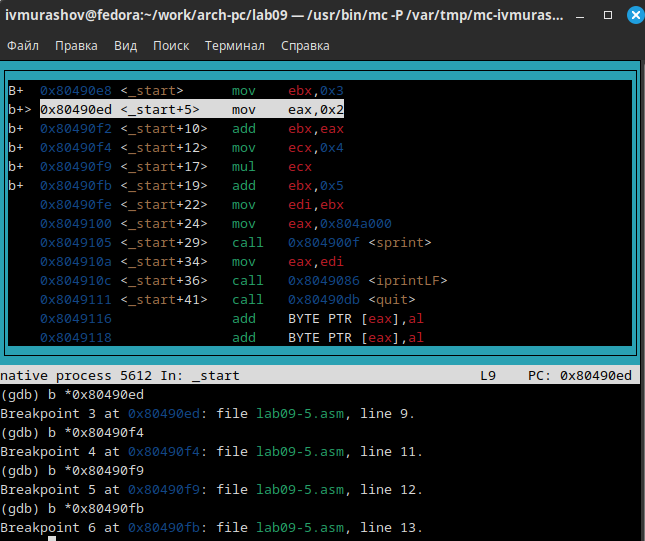
Запускаю программы. рис. [??]).



Запуск программы в оболочке GDB

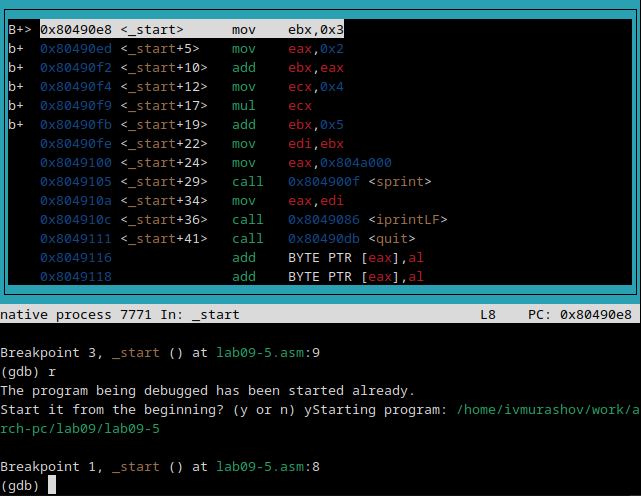
В результате выводится ‘10’, а должно получиться ‘25’. Следовательно, программа работает некорректно.

Устанавливаю брейкпоинты во всех местах, где происходит вычисление значений (рис. [??]).



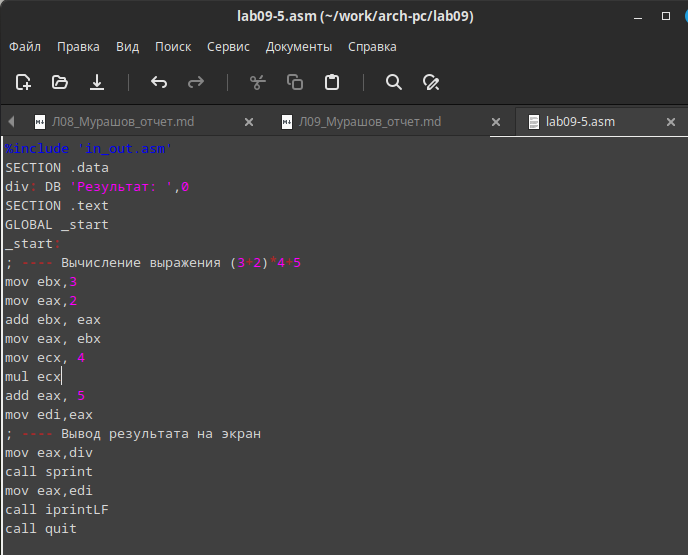
Установка брейкпоинтов в оболочке GDB

С помощью команды continue (сокращённо ‘c’) просматриваю значения и нахожу ошибки (рис. [??]).



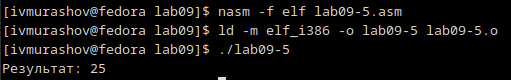
Работа с переменными в оболочке GDB

При выполнении инструкции mul значение регистра ecx умножается на значение регистра eax. Добавляю инструкцию ‘mov eax, ebx’. Изменяю инструкцию ‘add ebx, 5’ на ‘add eax, 5’ и ‘mov edi,ebx’ на ‘mov edi,eax’ (рис. [??]).



Редактирование файла

Создаю исполняемый файл и запускаю его. Проверяю работу программы, вручную посчитав искомое значение. Результаты совпали (рис. [??]).



Трансляция, компоновка и запуск файлов

# 4 Выводы

В ходе выполнения данной лабораторной работы я приобрёл навыки написания программ с использованием подпрограмм, познакомился с методами отладки при помощи GDB и его основными возможностями.