Отчёт по лабораторной работе №9

Выполнил студент НКАбд-02-25

Арина Андреевна Дрекина

Содержание

1 Цель работы. 3

2 Порядок выполнения лабораторной работы. 4

3 Реализация подпрограмм в NASM. 5

4 Отладка программам с помощью GDB. 11

5 Добавление точек останова. 18

6 Работа с данными программы в GDB. 20

7 Обработка аргументов командной строки в GDB. 26

8 Задание для самостоятельной работы. 30

9 Вывод. 34

1 Цель работы.

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

2 Порядок выполнения лабораторной работы.

3 Реализация подпрограмм в NASM.

Я создала каталог для выполнения лабораторной работы №9, затем перешла в него и создала текстовый файл lab09-1.asm. (Рисунок 3.1)

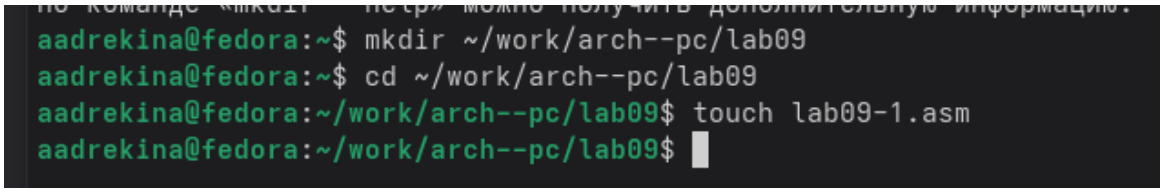

Рисунок 3.1: Создание каталога для лабораторной работы.

Я ввела в файл текст из Листинга 9.1. Перед этим изучила программу (Рисунок 3.2)

Открыть

lab09-1.asm
~/work/arch--pc/lab09

.md | COURSE | Makefile | ЛО4_Дрекина_ | ЛО7_Дрекина_

```
%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
```

Рисунок 3.2: Ввод программы в файл.

Потом я запустила программу. (Рисунок 3.3)

```
aadrekina@fedora:~/work/arch--pc/lab09$ nasm -f elf lab09-1.asm
aadrekina@fedora:~/work/arch--pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
aadrekina@fedora:~/work/arch--pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
aadrekina@fedora:~/work/arch--pc/lab09$
```

Рисунок 3.3: Запуск.

Листинг 9.1:

```
%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
```

Потом я изменила текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul, чтобы вычислялось выражение f(g(x)), где x вводится с клавиатуры, f(x)=2*x*+7, *g(x)*=3x-1 . (Рисунок 3.4)

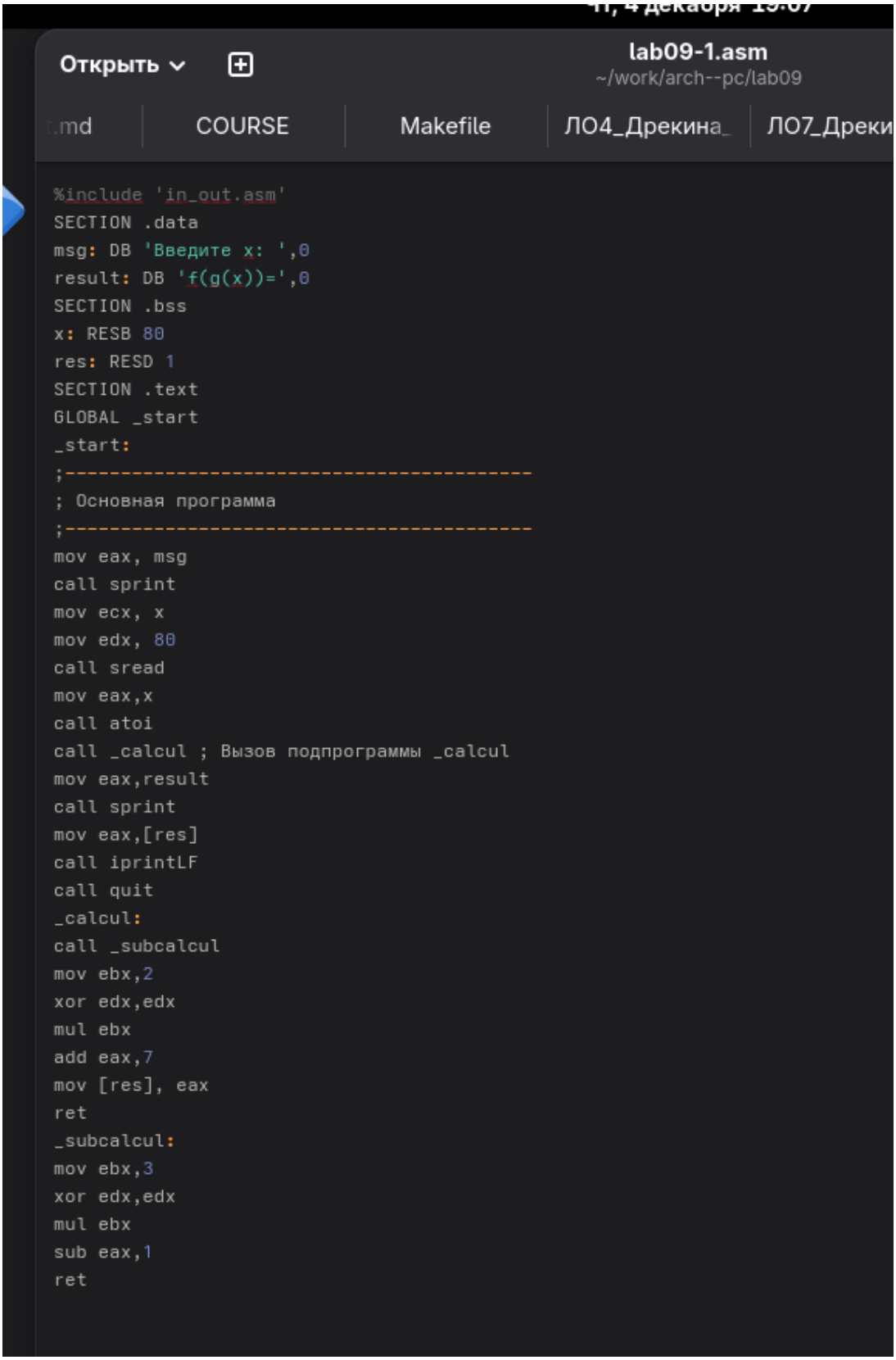

Рисунок 3.4: Изменения в программе.

Затем я сделала файл исполняемым и запустила программу. (Рисунок 3.5)

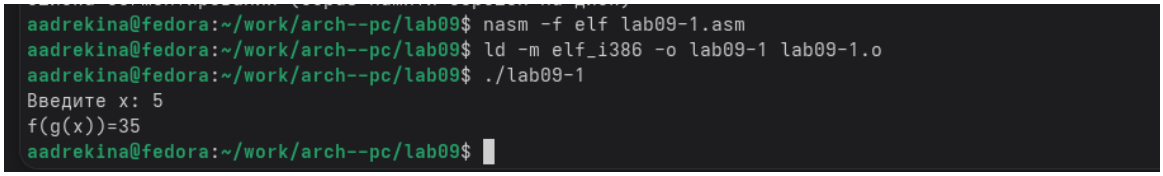

Рисунок 3.5: Запуск программы.

Программа посчитала все правильно и я продолжила работу дальше.

4 Отладка программам с помощью GDB.

Затем я создала еще один текстовый файл и ввела текст из Листинга 9.2. (Рисунок 4.1)

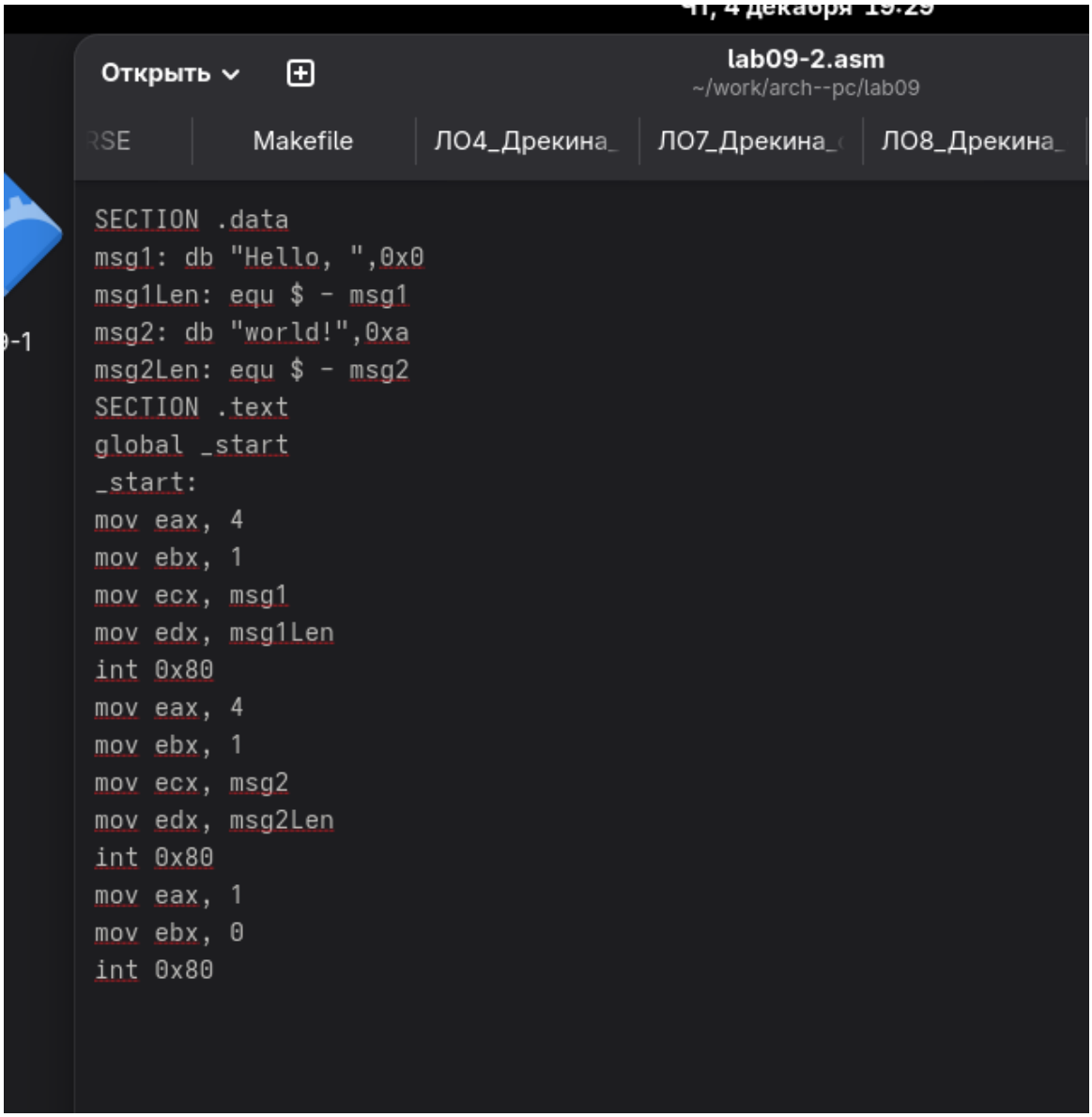

Рисунок 4.1: Ввод текста из Листинга 9.2.

Листинг 9.2:

```
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
```

Потом я сделала этот файл исполняемым, но для работы с GDB в исполняемый файл я добавила отладочную информацию, для этого трансляцию программы я провела с ключом «-g». (Рисунок 4.2)

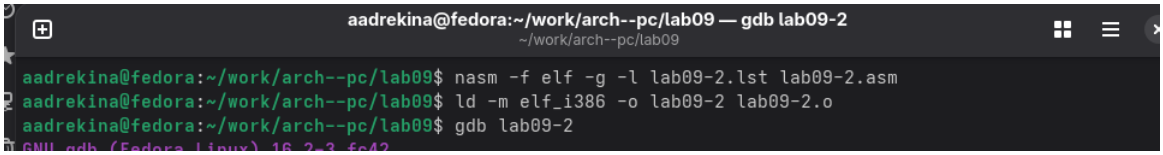

Рисунок 4.2: Создание исполняемого файла с ключом „-g“.

Затем я загрузила исполняемый файл в отладчик gdb. (Рисунок 4.3)

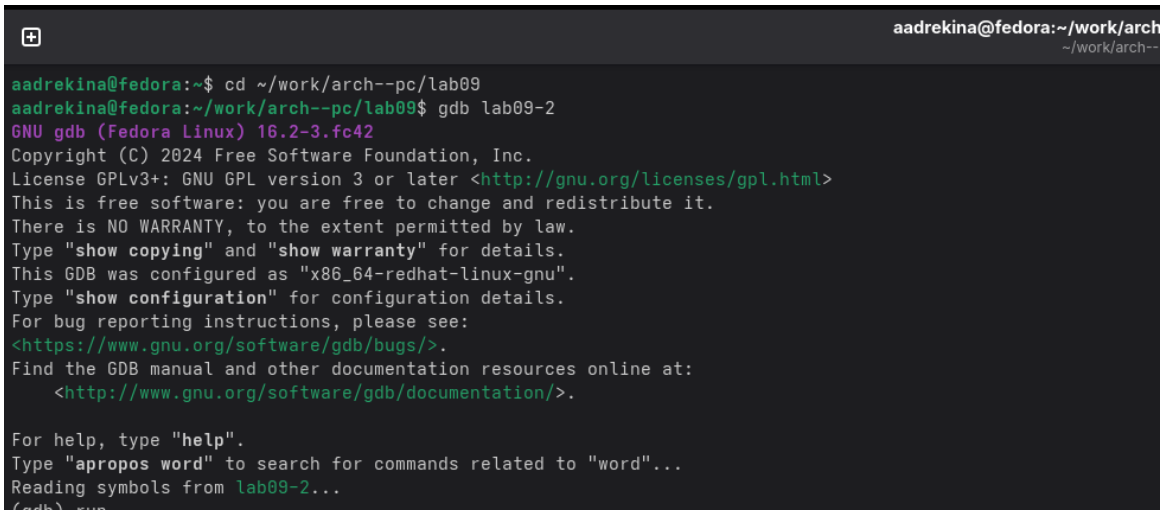

Рисунок 4.3: Загрузка исполняемого файла.

Затем, чтобы проверить работу программы, я запустила эту программу в оболочке GDB с помощью команды run. (рисунок 4.4)

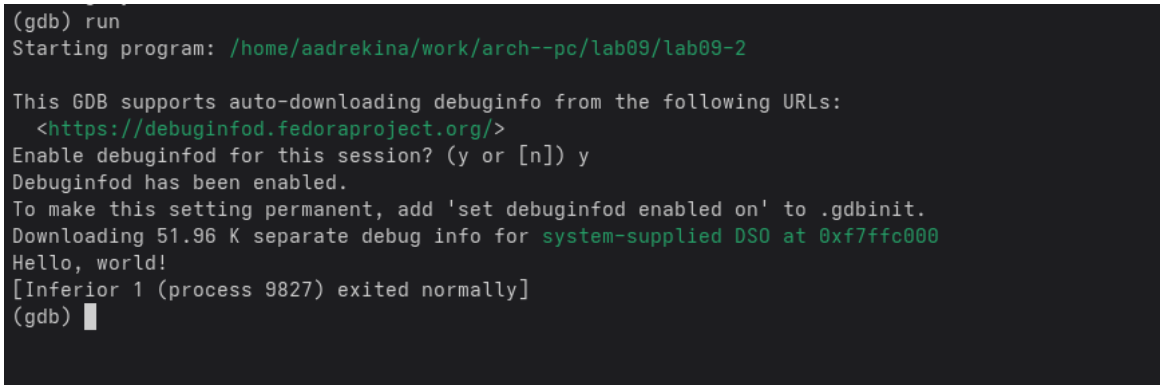

Рисунок 4.4: Запуск программы в оболочке GDB.

Затем, чтобы более подробно проанализировать эту программу я установила брейк-поинт на метку _start. (Рисунок 4.5)

Рисунок 4.5: Установка брейкпоинта.

Затем я посмотрела дисассимилированный код программы с помощью команды disassemble начиная с метки _start. (Рисунок 4.6)

Рисунок 4.6: Просмотр дисассимилированного кода программы.

Затем я переключилась на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel. (Рисунок 4.7)

Рисунок 4.7: Переключение на отображение команд с Intel’овским синтаксисом.

Различия отображения синтаксиса машинных команд в режимах ATT и Intel заключается в том, что в Intel синтаксические операнды идут в порядке dst,src и еще регистры пишутся без %. А в ATT синтаксис обратный, сначала src, а потом dst и при этом используется % для регистров и $ для констант.

Затем я включила режим псевдографики, чтобы анализировать программы было удобнее. (Рисунок 4.8)

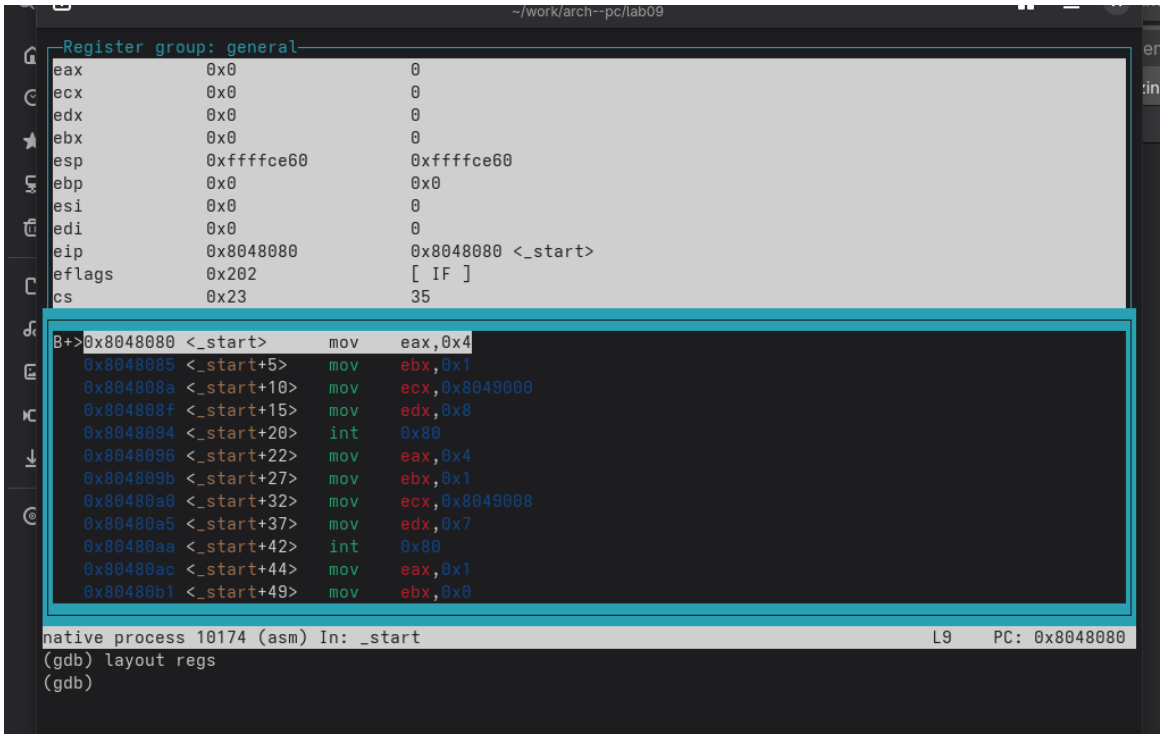

Рисунок 4.8: Включение режима псевдографики.

5 Добавление точек останова.

С помощью команды info breakpoints (кратко i b) я проверила установлена ли точка останова. (Рисунок 5.1)

Рисунок 5.1: Проверка точки останова.

Затем я установила еще одну точку останова по адресу инструкции. Адрес я взяла у предпоследней инструкции (mov ebx, 0x0) (Рисунок 5.2)

Рисунок 5.2: Установление еще одной точки останова.

Затем я еще раз ввела команду info breakpoints, чтобы посмотреть информацию о всех установленных точках останова. (Рисунок 5.3)

Рисунок 5.3: Проверка точек останова.

6 Работа с данными программы в GDB.

Я выполнила 5 инструкций с помощью команды stepi(или si). (Рисунок 6.1), (Рисунок 6.2),(Рисунок 6.3),(Рисунок 6.4),(Рисунок 6.5)

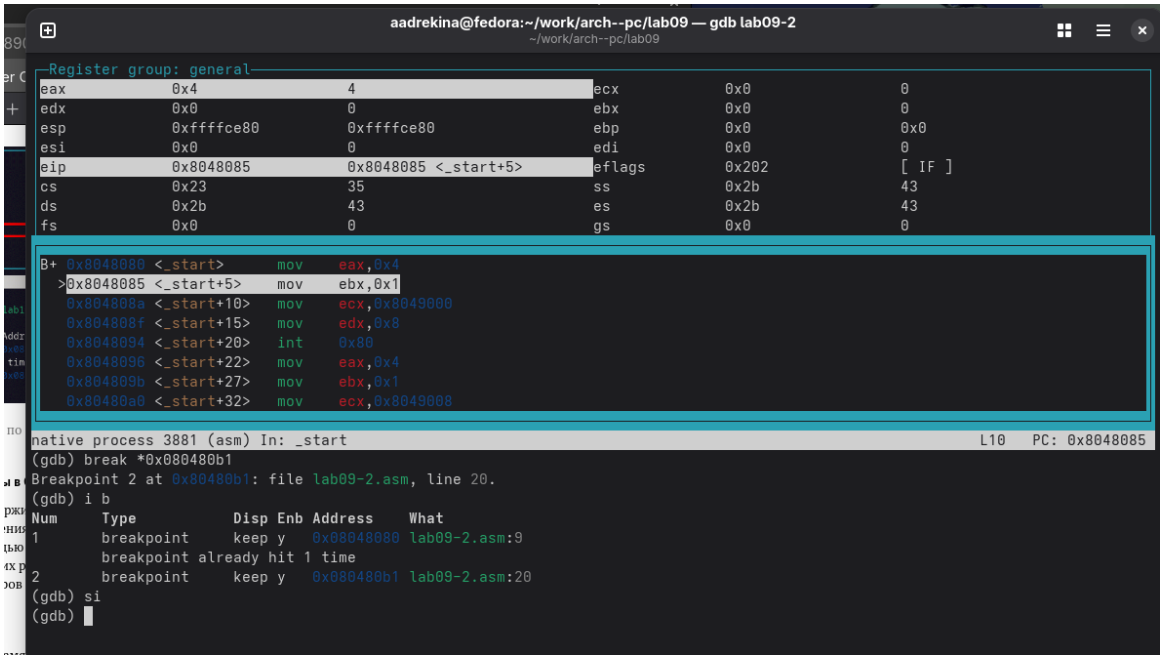

Рисунок 6.1: Первая инструкция.

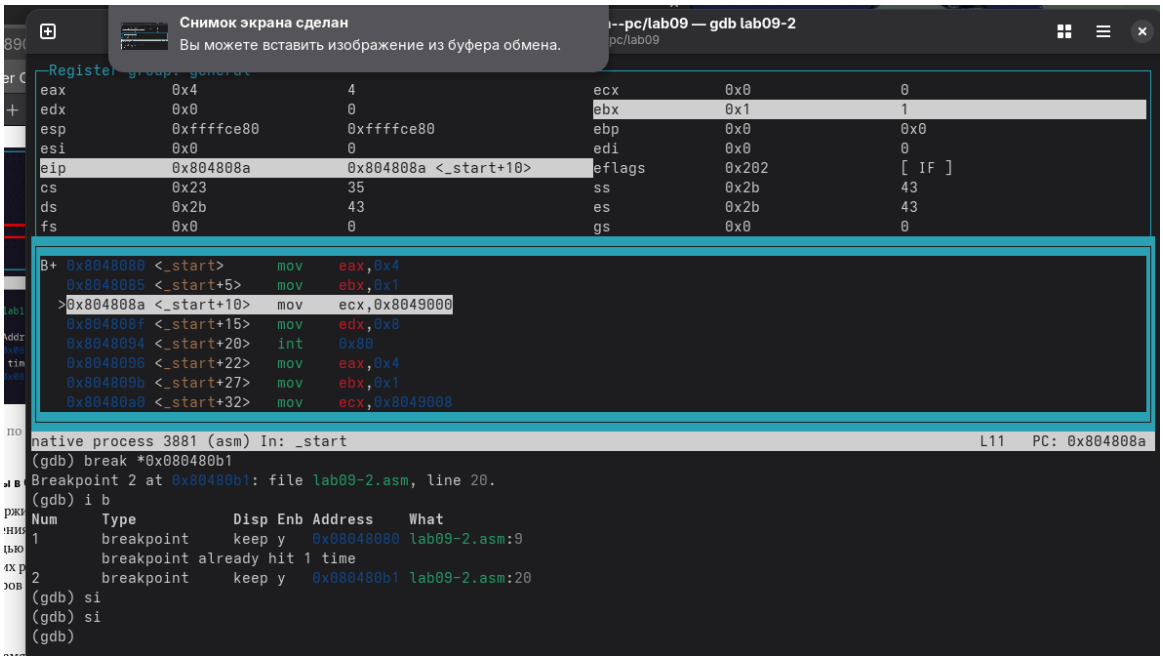

Рисунок 6.2: Вторая инструкция.

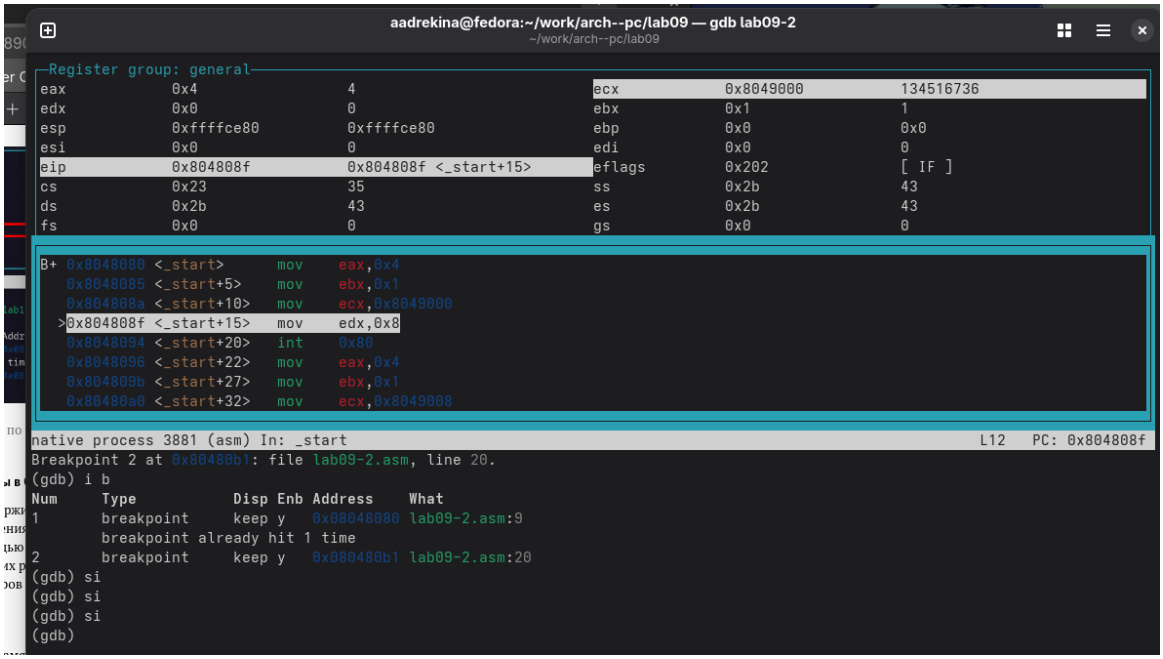

Рисунок 6.3: Третья инструкция.

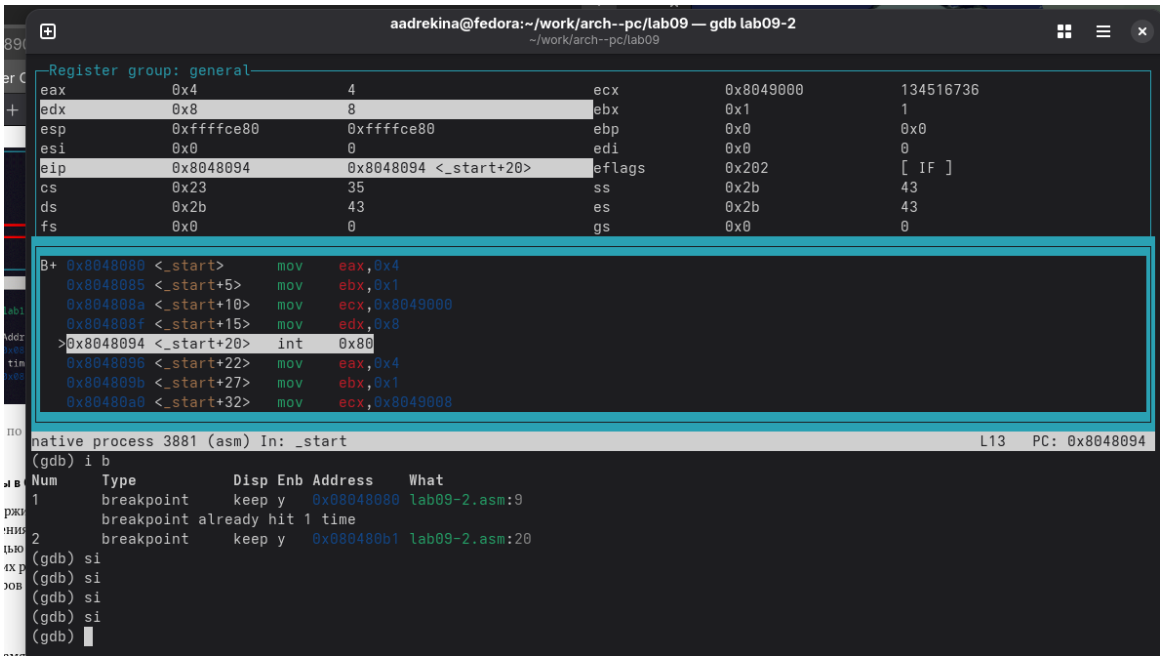

Рисунок 6.4: Четвертая инструкция.

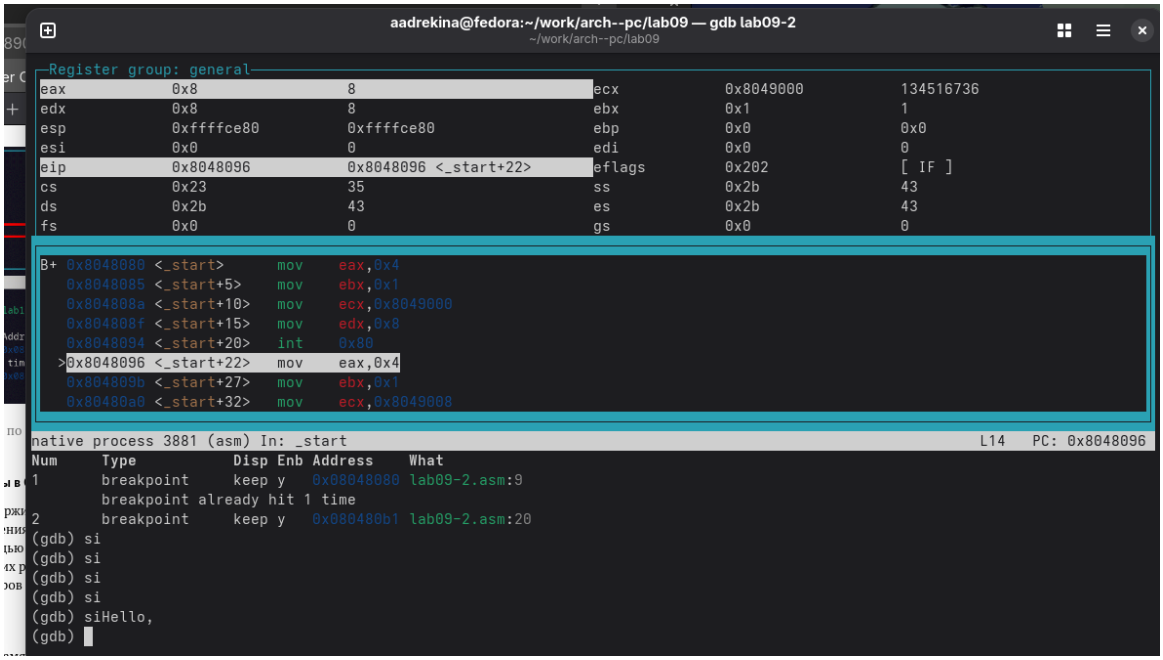

Рисунок 6.5: Пятая инструкция.

Я внимательно изучила эти 5 скринов, и проследила за изменением регистров. Меняются такие регистры как: EAX,EBX,ECX,EDX.

Я посмотрела содержимое регистров с помощью команды info registers. (Рисунок 6.6)

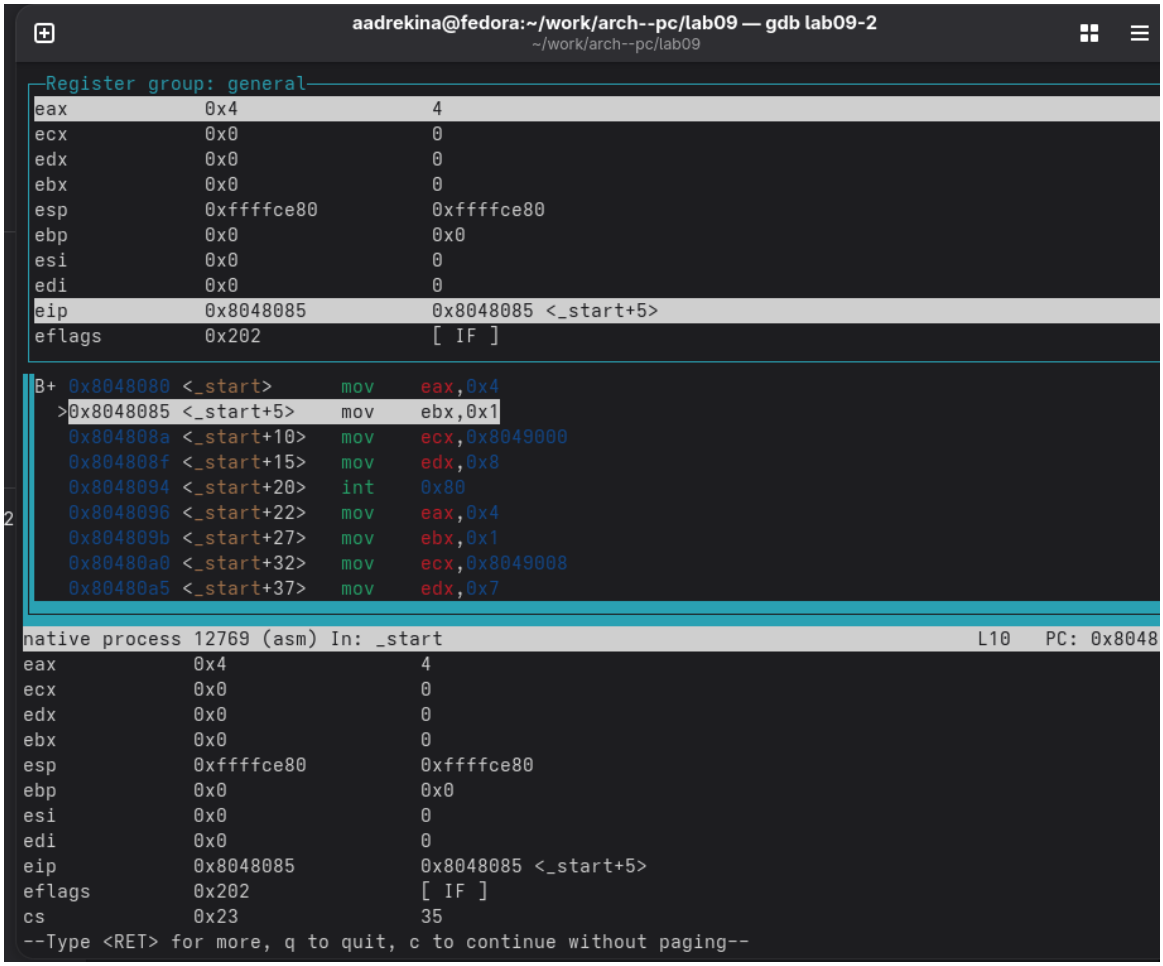

Рисунок 6.6: Просмотр содержимого регистра.

Потом я посмотрела значение переменной msg1 по имени. (Рисунок 6.7)

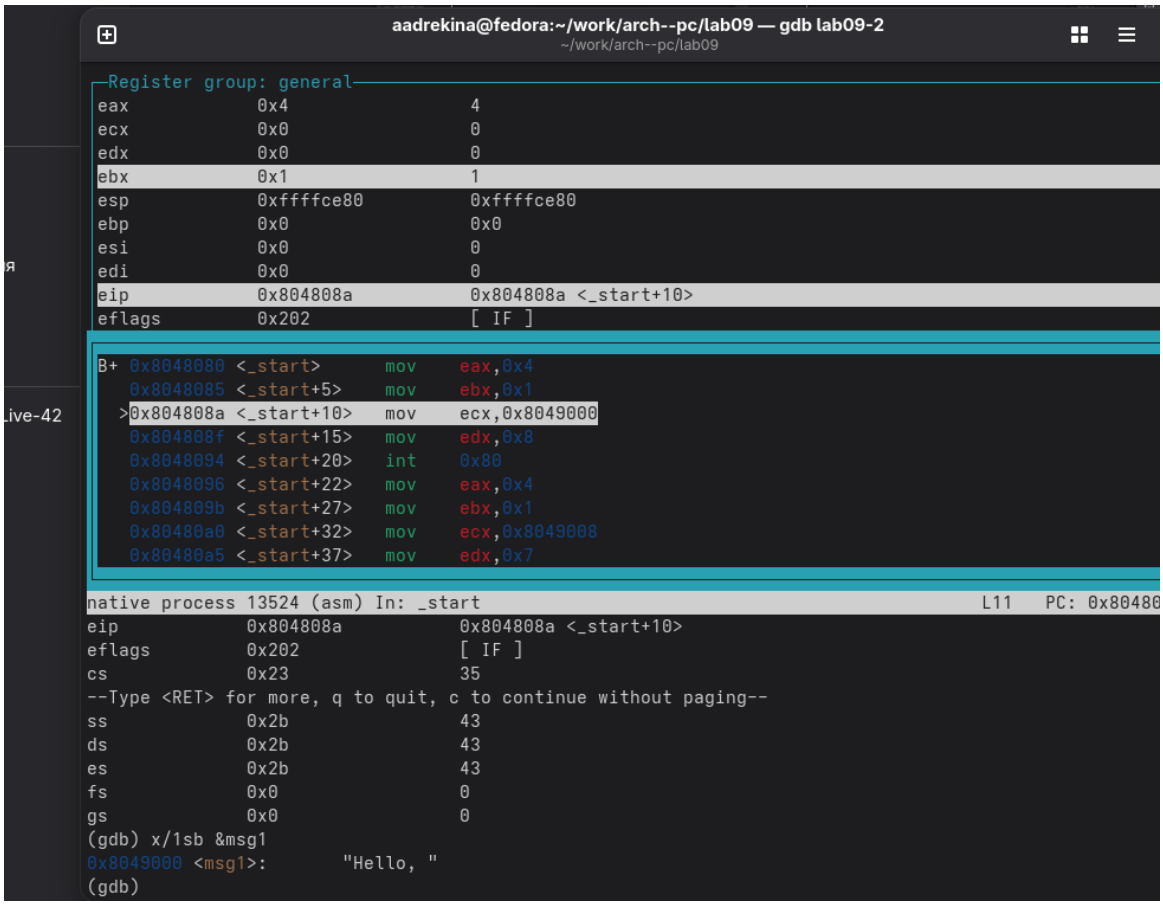

Рисунок 6.7: Значение переменной msg1.

У меня вывелся такой же ответ, как и в лекции, значит я сделала все правильно.

Потом, таким же образом я посмотрела значение переменной msg2 по адресу. (Рисунок 6.8)

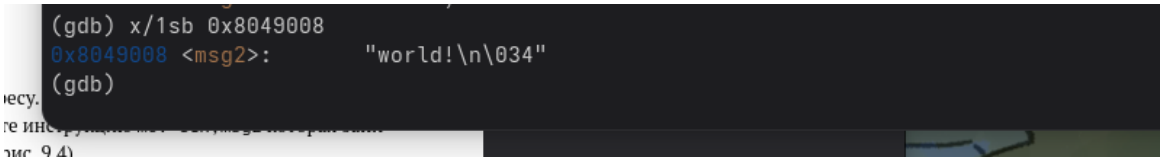

Рисунок 6.8: Значение переменной msg2.

Затем я изменила значение для регистра с помощью команды set. (Рисунок 6.9)

Рисунок 6.9: Изменение значений для регистра msg1.

Ответ вывелся такой же, значит я сделал все правильно.

Затем я также изменила значение для регистра msg2. (Рисунок 6.10)

Рисунок 6.10: Изменения значений для регистра msg2.

Затем с помощью команды set я изменила значение регистра. (Рисунок 6.11)

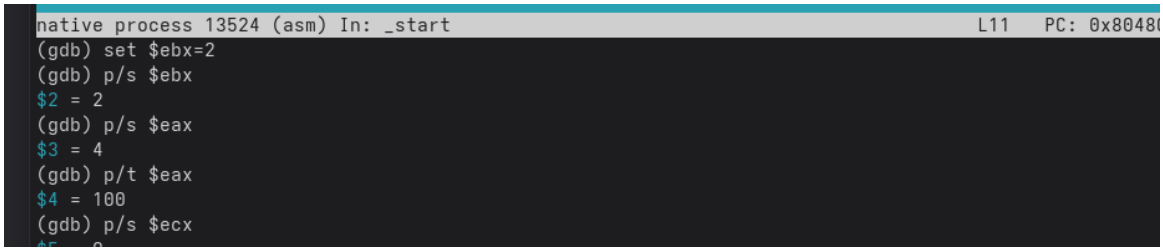

Рисунок 6.11: Изменение значение регистра.

Команда p/s $ebx выводит строку по адресу, хранящемуся в ebx, следовательно при set $ebx=«2» показывается ASCLL-код символа, а при set $ebx=2 - числовое значение. Команда p/t $ebx выводит значение регистра в двоичном коде.

В конце я завершила выполнение программы с помощью команды continue и вышла из GDB с помощью команды quit.

7 Обработка аргументов командной строки в GDB.

Я скопировала файл lab8-2.asm, который был создан при выполнении лабораторной работы №8 в файл lab09-3.asm. Затем я создала исполняемый файл (Рисунок 7.1)

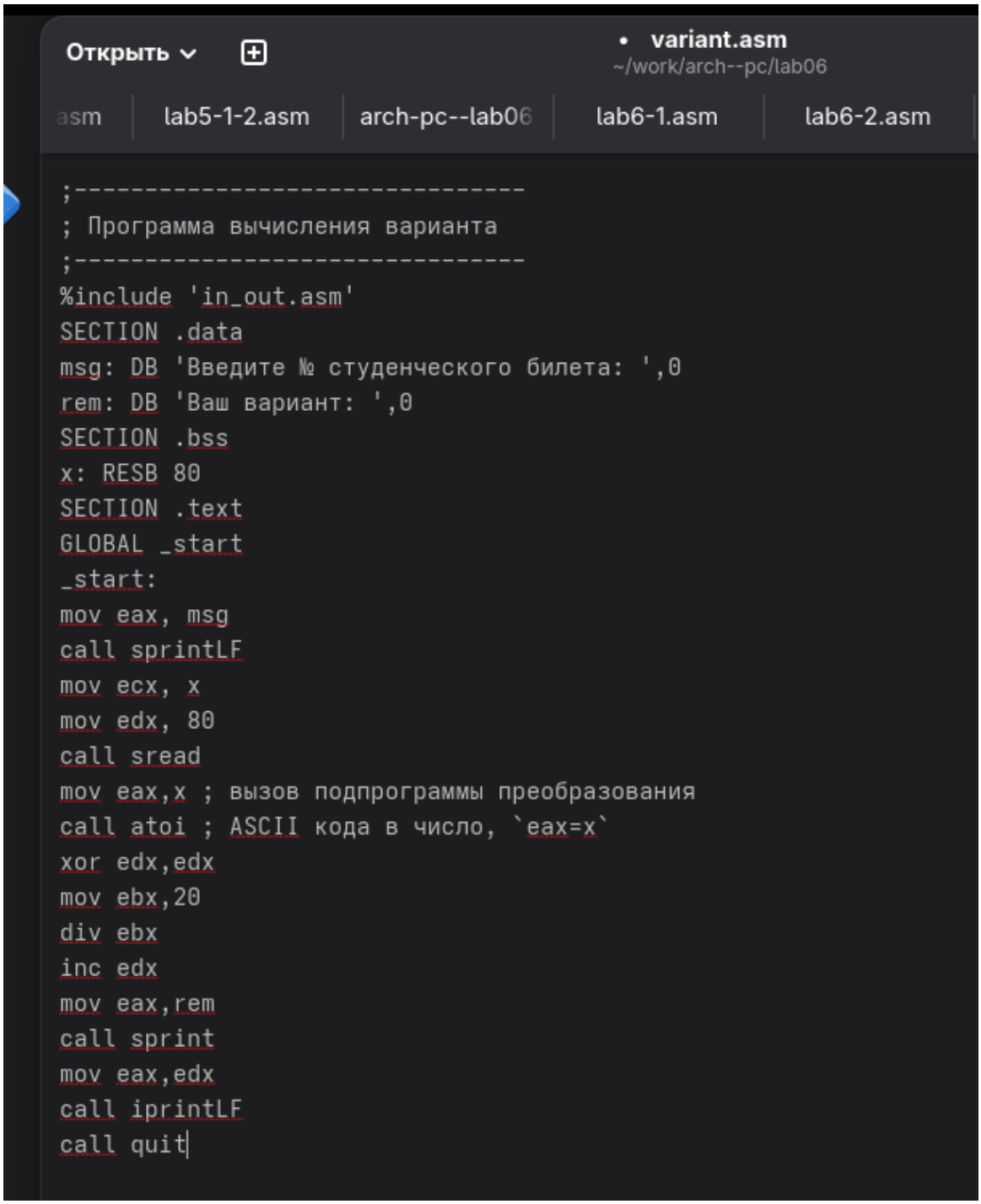

Рисунок 7.1: Копирование файла.

Чтобы загрузить в GDB программы с аргументами необходимо использовать ключ –args (Рисунок 7.2)

```
aadrekina@fedora:~/work/arch--pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Fedora Linux) 16.2-3.fc42
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
```

Рисунок 7.2: Загрузка исполняемого файла в отладчик, указав аргументы.

Потом я установила точку останова перед первой инструкцией и запустила ее. (Рисунок 7.3)

```
(gdb) b _start
Breakpoint 1 at 0x8048148: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/aadrekina/work/arch--pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Breakpoint 1, _start () at lab09-3.asm:5
5       pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рисунок 7.3: Установка точки останова и запуск.

Я посмотрела позиции стека - по адресу [esp+4].(Рисунок 7.4)

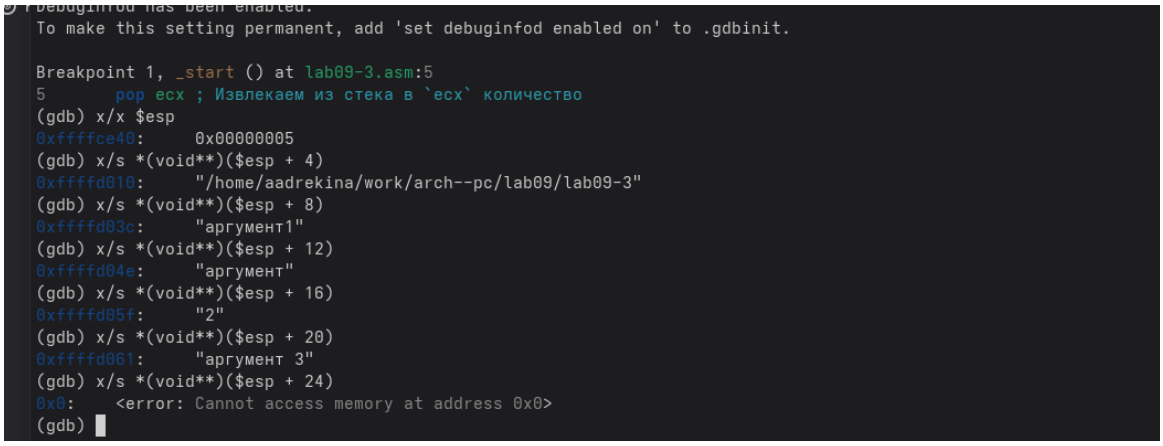

Рисунок 7.4: Просмотр остальных позиций стека.

Шаг изменения равен 4,так как в данной программе каждый адрес в памяти занимает 4 байта, стек хранит обычный список адресов, и для того чтобы перейти к следующему адресу нужно сдвинутся ровно на 4 байта.

8 Задание для самостоятельной работы.

Я преобразовала программу созданную во время выполнения лабораторной работы №8, реализовав вычисления функции f(x) как подпрограмму. (Рисунок 8.1)

Рисунок 8.1: Преобразования программы.

Я создала еще один текстовый файл и вставила туда Листинг 9.3. Я начала анализировать эту программу с помощью отладчика GDB, определила ошибки и исправила их. (Рисунок 8.2)

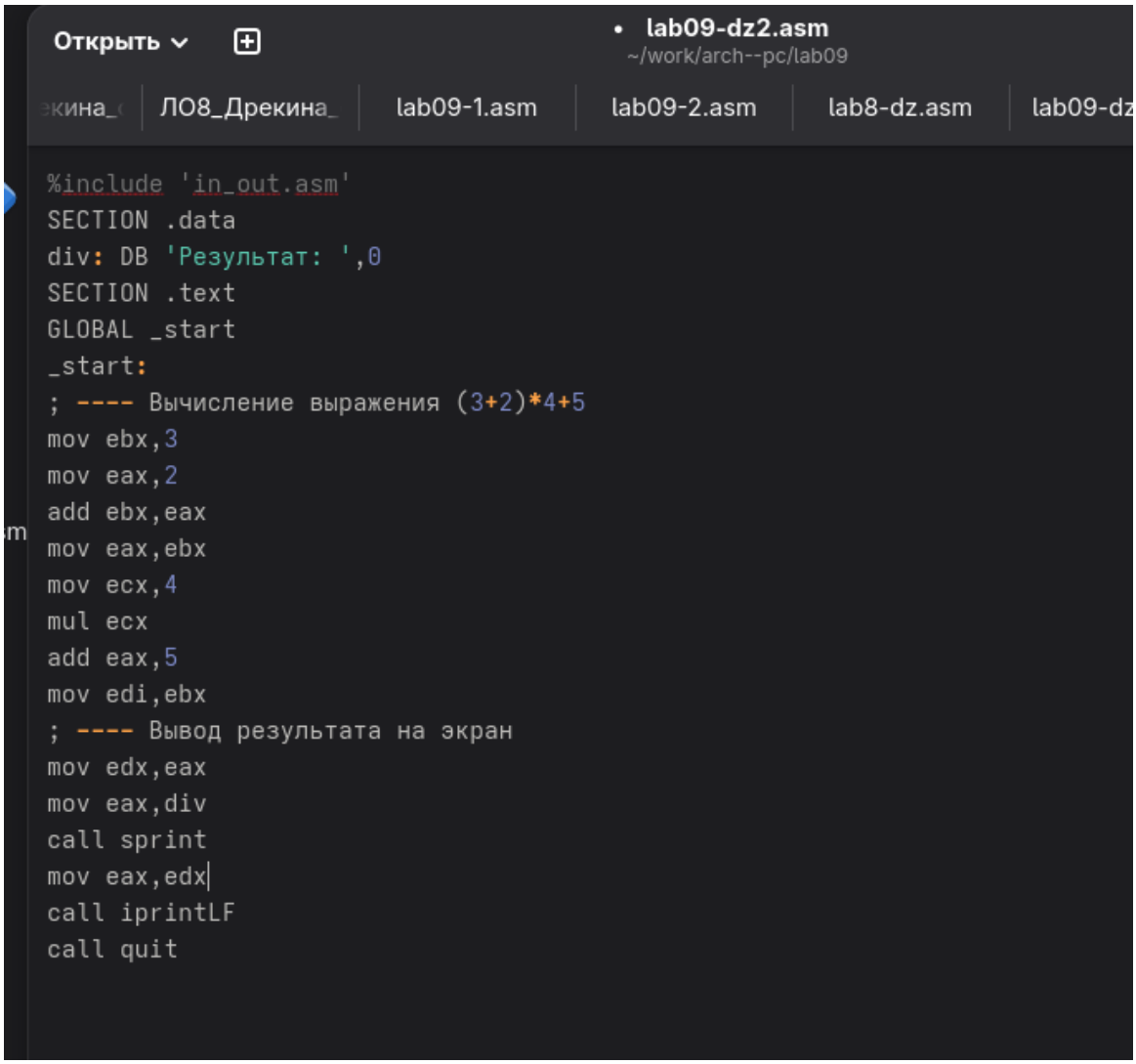

Рисунок 8.2: Изменение в программе.

Листинг 9.3:

```
%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
```

Затем я сделала файл исполняемым и проверила результат, ответы совпали. Значит я правильно изменила программу.

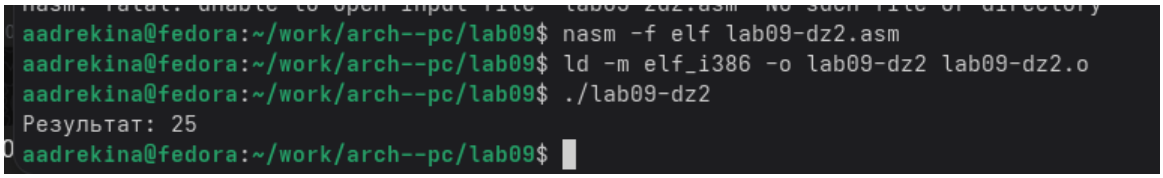

Рисунок 8.3: Запуск.

9 Вывод.

Я приобрела навыки написания программы с использованием подпрограммы. И ознакомилась с методами отладки при помощи GDB и его основными возможностями.