

# **Отчет по лабораторной работе №9**

**дисциплина: Архитектура компьютера**

Маньковская Дарья Станиславовна

# Содержание

## **Список иллюстраций**

# Список таблиц

## 1 Цель работы

Цель данной лабораторной работы - это приобретение практического опыта в написании программ с использованием подпрограмм, а также знакомство с методами отладки при помощи gdb и его основными возможностями.

## 2 Задание

1. Реализация подпрограмм в NASM.
2. Отладка программ при помощи gdb.
3. Выполнение заданий для самостоятельной работы

## 3 Теоретическое введение

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, отработывает, 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). Вторым этапом — 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 местонахож-

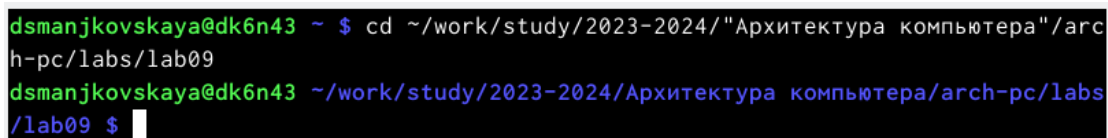
дения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново. Наиболее часто применяют следующие методы отладки: • создание точек контроля 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. GDB (GNU Debugger — отладчик проекта GNU) [1] 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, неко-

торые 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

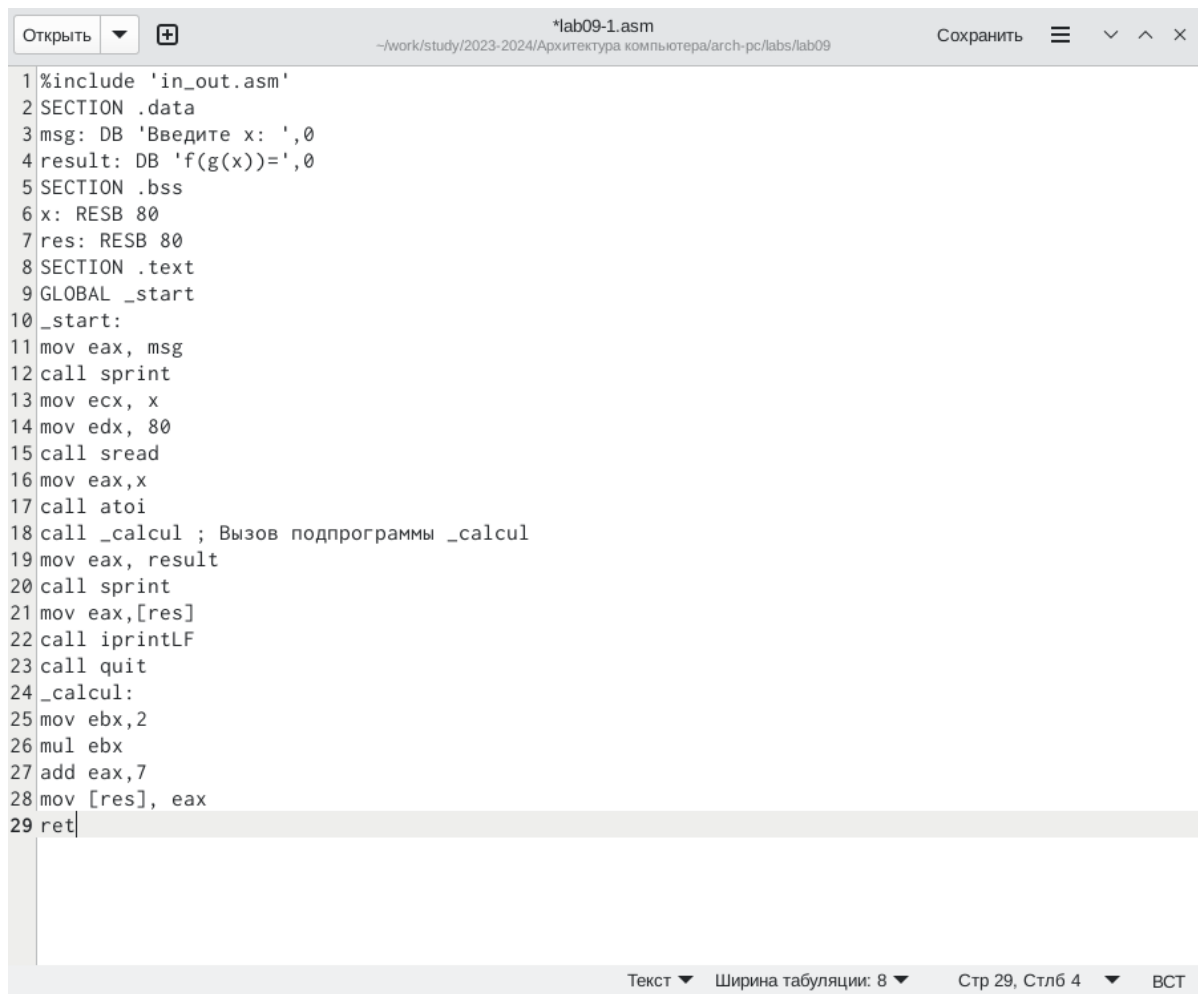
- 1) Реализация подпрограмм в NASM Перехожу в каталог, созданный для файлов с программами для лабораторной работы №9. С помощью touch создаю файл lab09-1.asm. Копирую в текущий каталог файл in\_out.asm, так как он будет использоваться в дальнейшем (рис. 1).



```
dsmanjovskaya@dk6n43 ~ $ cd ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab09
dsmanjovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $
```

Рис. 1: Перемещение между директориями, создание файла

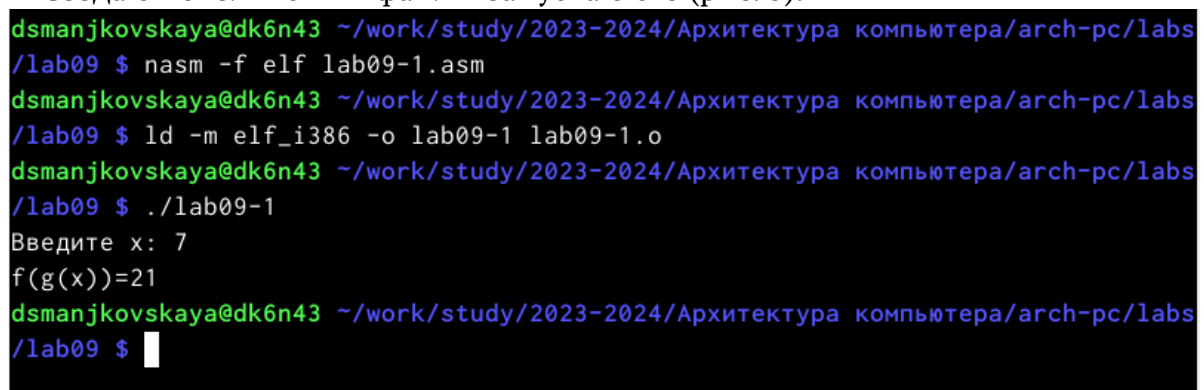
Открываю созданный файл lab09-1.asm, вставляю в него программу из листинга 9.1 (рис. 2)



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB 'f(g(x))=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 mov eax, msg
12 call sprint
13 mov ecx, x
14 mov edx, 80
15 call sread
16 mov eax, x
17 call atoi
18 call _calcul ; Вызов подпрограммы _calcul
19 mov eax, result
20 call sprint
21 mov eax, [res]
22 call iprintLF
23 call quit
24 _calcul:
25 mov ebx, 2
26 mul ebx
27 add eax, 7
28 mov [res], eax
29 ret
```

Рис. 2: Редактирование файла

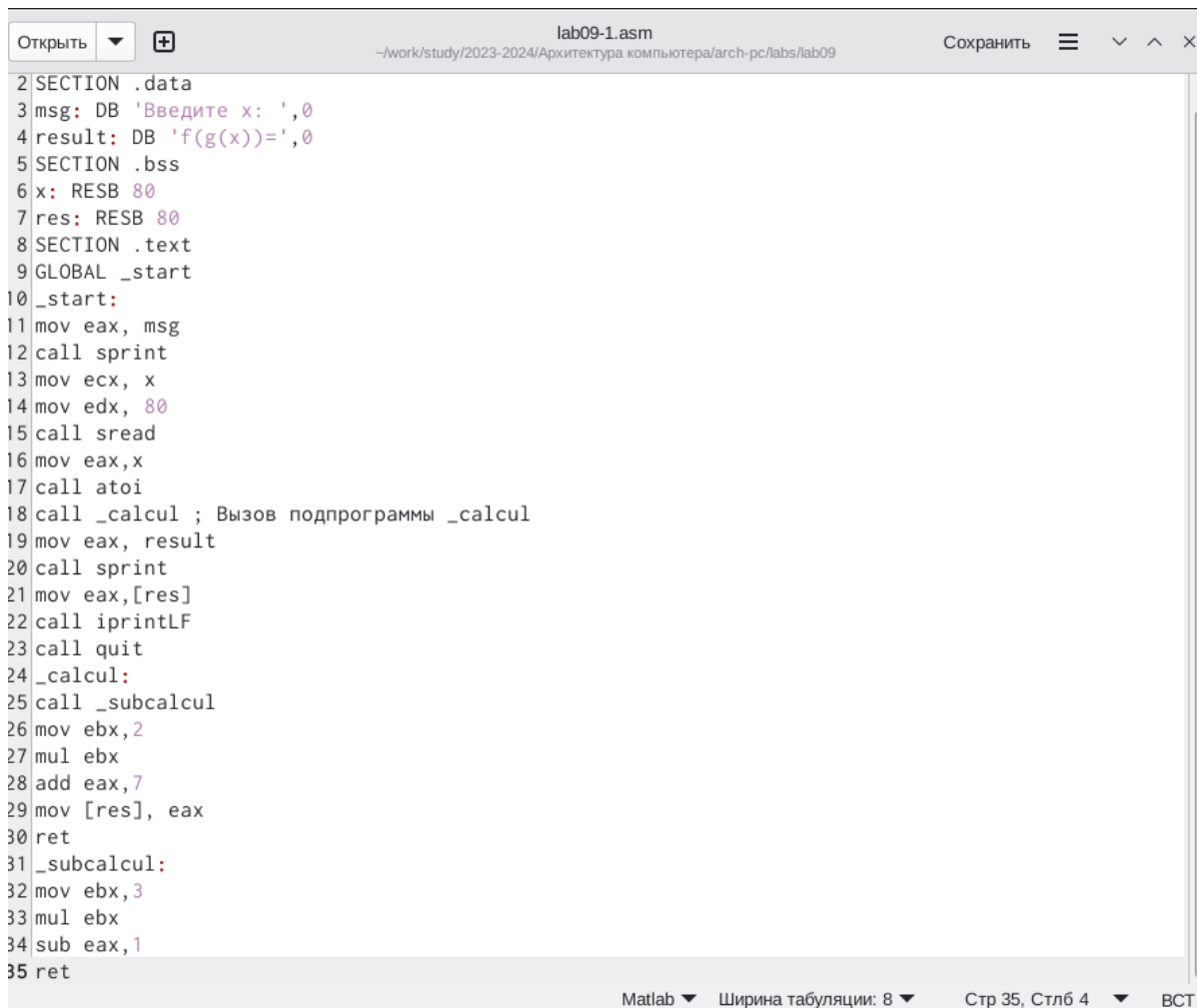
Создаю исполняемый файл и запускаю его (рис. 3).



```
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs
/lab09 $ nasm -f elf lab09-1.asm
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs
/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs
/lab09 $ ./lab09-1
Введите x: 7
f(g(x))=21
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs
/lab09 $
```

Рис. 3: Запуск исполняемого файла

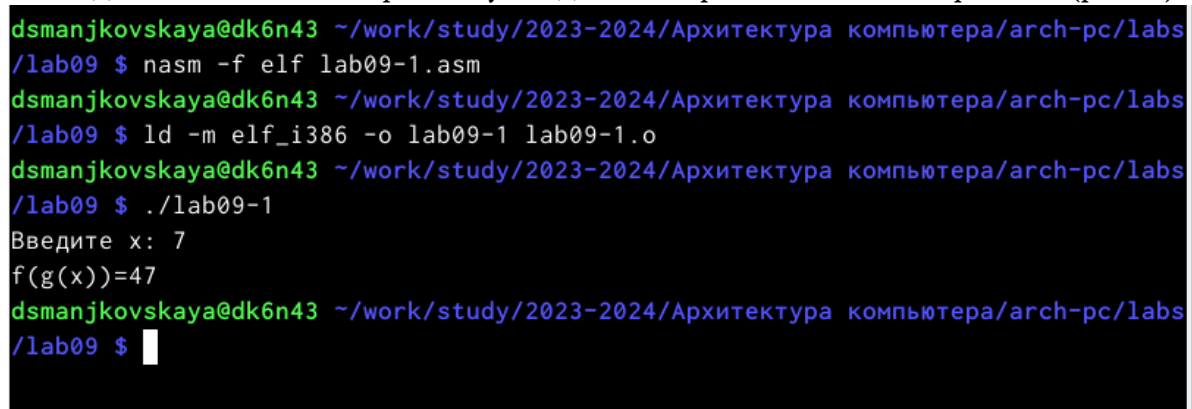
Добавляю подпрограмму `subcalcul_`, чтобы программа вычисляла значение  $f(g(x))$  (рис. 4).



```
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB 'f(g(x))=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 mov eax, msg
12 call sprint
13 mov ecx, x
14 mov edx, 80
15 call sread
16 mov eax, x
17 call atoi
18 call _calcul ; Вызов подпрограммы _calcul
19 mov eax, result
20 call sprint
21 mov eax, [res]
22 call iprintLF
23 call quit
24 _calcul:
25 call _subcalcul
26 mov ebx, 2
27 mul ebx
28 add eax, 7
29 mov [res], eax
30 ret
31 _subcalcul:
32 mov ebx, 3
33 mul ebx
34 sub eax, 1
35 ret
```

Рис. 4: Редактирование файла

Создаю исполняемый файл и убеждаюсь в правильности его работы (рис. 5).



```
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09
$ nasm -f elf lab09-1.asm
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09
$ ld -m elf_i386 -o lab09-1 lab09-1.o
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09
$ ./lab09-1
Введите x: 7
f(g(x))=47
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09
$
```

Рис. 5: Запуск исполняемого файла

## 2) Отладка программ при помощи gdb

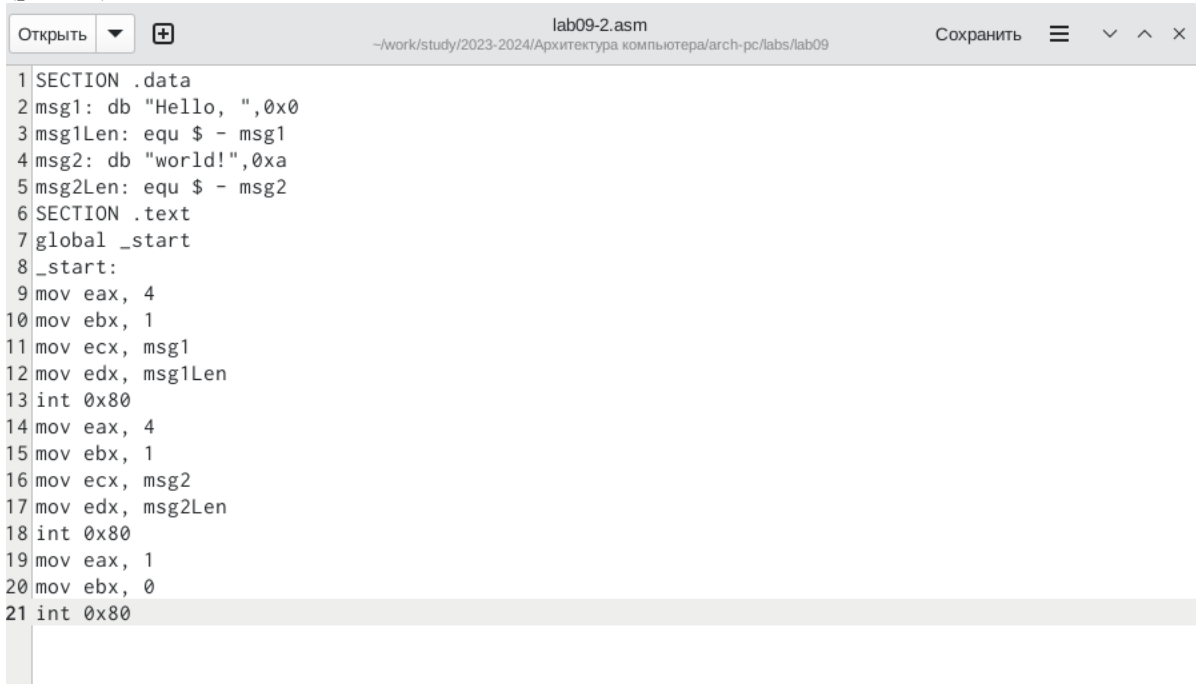


Создаю файл lab09-2.asm с помощью утилиты touch (рис. 6).

```
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ touch lab09-2.asm
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $
```

Рис. 6: Создание файла

Открываю созданный файл и вношу в него текст программы из листинга 9.2 (рис. 7)



```
lab09-2.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09
Сохранить

1 SECTION .data
2 msg1: db "Hello, ",0x0
3 msg1Len: equ $ - msg1
4 msg2: db "world!",0xa
5 msg2Len: equ $ - msg2
6 SECTION .text
7 global _start
8 _start:
9 mov eax, 4
10 mov ebx, 1
11 mov ecx, msg1
12 mov edx, msg1Len
13 int 0x80
14 mov eax, 4
15 mov ebx, 1
16 mov ecx, msg2
17 mov edx, msg2Len
18 int 0x80
19 mov eax, 1
20 mov ebx, 0
21 int 0x80
```

Рис. 7: Редактирование файла

Создаю исполняемый файл (рис. 8).

```
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $
```

Рис. 8: Запуск исполняемого файла

Загружаю файл в отладчик gdb (рис. 9).

```

dsmanjkovskaya@dk6n43 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs
/lab09 $ gdb lab09-2
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab09-2...
(gdb)

```

Рис. 9: Отладчик

Запускаю программу с помощью команды `run` и убеждаюсь в корректности работы программы (рис. 10)

```

(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/s/dsmanjkovskaya/work/study/202
3-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-2
Hello, world!
[Inferior 1 (process 793352) exited normally]
(gdb)

```

Рис. 10: Запуск программы

Устанавливаю метку `_start` и запускаю програаму. Вижу работу метки (рис. 11).

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/s/dsmanjkovskaya/work/study/202
3-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)

```

Рис. 11: Установка и работа метки

Смотрю дисассимилированный код программы с помощью команды

disassemble начиная с метки `_start` (рис. 12)

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4,%eax
    0x08049005 <+5>:      mov     $0x1,%ebx
    0x0804900a <+10>:     mov     $0x804a000,%ecx
    0x0804900f <+15>:     mov     $0x8,%edx
    0x08049014 <+20>:     int     $0x80
    0x08049016 <+22>:     mov     $0x4,%eax
    0x0804901b <+27>:     mov     $0x1,%ebx
    0x08049020 <+32>:     mov     $0x804a008,%ecx
    0x08049025 <+37>:     mov     $0x7,%edx
    0x0804902a <+42>:     int     $0x80
    0x0804902c <+44>:     mov     $0x1,%eax
    0x08049031 <+49>:     mov     $0x0,%ebx
    0x08049036 <+54>:     int     $0x80
End of assembler dump.
(gdb) █
```

Рис. 12: Дисассимилированный код

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 13). Различия отображения синтаксиса машинных команд в режимах АТТ и Intel: в синтаксисе АТТ перед именем регистра ставится префикс `%`, адрес инструкции пишется перед именем регистра (в Intel - наоборот), перед ним ставится префикс `$`.

```

(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Undefined command: "disassemble". Try "help".
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     eax,0x4
    0x08049005 <+5>:      mov     ebx,0x1
    0x0804900a <+10>:     mov     ecx,0x804a000
    0x0804900f <+15>:     mov     edx,0x8
    0x08049014 <+20>:     int     0x80
    0x08049016 <+22>:     mov     eax,0x4
    0x0804901b <+27>:     mov     ebx,0x1
    0x08049020 <+32>:     mov     ecx,0x804a008
    0x08049025 <+37>:     mov     edx,0x7
    0x0804902a <+42>:     int     0x80
    0x0804902c <+44>:     mov     eax,0x1
    0x08049031 <+49>:     mov     ebx,0x0
    0x08049036 <+54>:     int     0x80
End of assembler dump.
(gdb) █

```

Рис. 13: Дисассимилированный код программы с синтаксисом intel

Включая режим псевдографики (рис. 14) и (рис. 15)

```
Терминал - dsmanjkovskaya@dk6n43:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09
Файл  Правка  Вид  Терминал  Вкладки  Справка

B+> 0x8049000 <_start>      mov    eax,0x4
      0x8049005 <_start+5>    mov    ebx,0x1
      0x804900a <_start+10>   mov    ecx,0x804a000
      0x804900f <_start+15>   mov    edx,0x8
      0x8049014 <_start+20>   int     0x80
      0x8049016 <_start+22>   mov    eax,0x4
      0x804901b <_start+27>   mov    ebx,0x1
      0x8049020 <_start+32>   mov    ecx,0x804a008
      0x8049025 <_start+37>   mov    edx,0x7
      0x804902a <_start+42>   int     0x80
      0x804902c <_start+44>   mov    eax,0x1
      0x8049031 <_start+49>   mov    ebx,0x0
      0x8049036 <_start+54>   int     0x80

native process 805278 In: _start          L9      PC: 0x8049000
(gdb) 
```

Рис. 14: Режим псевдографики

```
[ Register Values Unavailable ]

B+> 0x8049000 <_start>      mov    eax,0x4
      0x8049005 <_start+5>    mov    ebx,0x1
      0x804900a <_start+10>   mov    ecx,0x804a000
      0x804900f <_start+15>   mov    edx,0x8
      0x8049014 <_start+20>   int     0x80
      0x8049016 <_start+22>   mov    eax,0x4
      0x804901b <_start+27>   mov    ebx,0x1

native process 805278 In: _start          L9      PC: 0x8049000
(gdb) layout regs
(gdb) 
```

Рис. 15: Режим псевдографики

Проверяю точки останова (рис. 16).

```
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
(gdb) █
```

Рис. 16: Проверка точек останова

Устанавливаю точку останова в последней инструкции (рис. 17)

```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) █
```

Рис. 17: Установка точки останова

Смотрю информацию обо всех установленных точках останова (рис. 18).

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
2        breakpoint     keep y   0x08049031 lab09-2.asm:20
(gdb) █
```

Рис. 18: Точки останова

Выполняю 5 инструкций `stepi`, и последовательно замечаю изменение значений регистров на экране соответственно (рис. 19)

```
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) █
```

Рис. 19: Изменение значений регистров

Просматриваю содержимое регистров с помощью команды `info registers` (рис.

20) и (рис. 21)

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc4c0 0xffffc4c0

0x804900a <_start+10> mov ecx,0x804a000
0x804900f <_start+15> mov edx,0x8
0x8049014 <_start+20> int 0x80
> 0x8049016 <_start+22> mov eax,0x4
0x804901b <_start+27> mov ebx,0x1
0x8049020 <_start+32> mov ecx,0x804a008
0x8049025 <_start+37> mov edx,0x7

native process 805278 In: _start L14 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc4c0 0xffffc4c0
ebp      0x0      0x0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 20: Просмотр содержимого регистров

```

cs      0x23      35
ss      0x2b      43
ds      0x2b      43
es      0x2b      43
--Type <RET> for more, q to quit, c to continue without paging--fs
x0      0
gs      0x0      0
(gdb)

```

Рис. 21: Просмотр содержимого регистров

Смотрю значение переменной msg1 по имени с помощью команды x & (рис. 22)

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

Рис. 22: Значение переменной msg1

Смотрю значение переменной msg2 по адресу (рис. 23).

```
(gdb) si
(gdb) x/1sb & msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) █
```

Рис. 23: Значение переменной msg2

Изменяю первый символ переменной msg1 с помощью команды set (рис. 24)

```
(gdb) set {char}&msg1 = 'h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) █
```

Рис. 24: Изменение переменной msg1

Аналогичные действия проделываю с переменной msg2 (рис. 25)

```
(gdb) set {char}&msg2 = 'W'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "World!\n\034"
(gdb) █
```

Рис. 25: Изменение переменной msg2

Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 26)



```

(gdb) p/s $edx
$1 = 8
(gdb) p/t $edx
$2 = 1000
(gdb) p/x $edx
$3 = 0x8
(gdb)

```

Рис. 26: Значения регистра edx

Изменяю значение регистра ebx с помощью команды set (рис. 27)

```

(gdb) set $ebx = '2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx = 2
(gdb) p/s $ebx
$5 = 2
(gdb)

```

Рис. 27: Изменение значений регистра

Разницу в выводе команд можно объяснить так: при бескавычном значении 2, мы получаем это же значение, а в другом случае переменная воспринимается иначе и на выходе мы видим значение 50.

Завершаю выполнение программы с помощью continue и выхожу из gdb с помощью quit.

Копирую файл lab8-2.asm, полученный во время выполнения лабораторной работы No8, содержащий программу для вывода аргументов командной строки. Затем создаю исполняемый файл (рис. 28)

```

dsmanjkovskaya@dk8n75 ~ $ cd ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab09
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ cp ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab09/lab09-3.asm
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ 

```

Рис. 28: Копирование файла, создание исполняемого файла

Загружаю исполняемый файл в отладчик, указав нужные аргументы (рис. 29)

```

dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ gdb --args lab09-3 argument1 argument2 'argument 3'
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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...
(gdb) 

```

Рис. 29: Загрузка файла в отладчик

Устанавливаю точку останова перед первой инструкцией и запускаю программу (рис. 30)

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/s/dsmanjkovskaya/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-3 argument1 argument2 'argument 3'

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

```

Рис. 30: Установка точки останова, запуск программы

Далее просматриваю позиции стека (рис. 31)

```

(gdb) x/s *(void**)(esp + 4)
0xfffffc47b:  "/afs/.dk.sci.pfu.edu.ru/home/d/s/dsmanjkovskaya/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc507:  "argument1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc511:  "argument"
(gdb) x/s *(void**)(esp + 16)
0xfffffc51a:  "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc51c:  "argument 3"
(gdb) x/s *(void**)(esp + 24)
0x0:  <error: Cannot access memory at address 0x0>
(gdb) 

```

Рис. 31: Просмотр позиций стека

Шаг изменения равен 4, т.к. каждый следующий адрес на стеке находится на расстоянии в 4 байта от предыдущего.

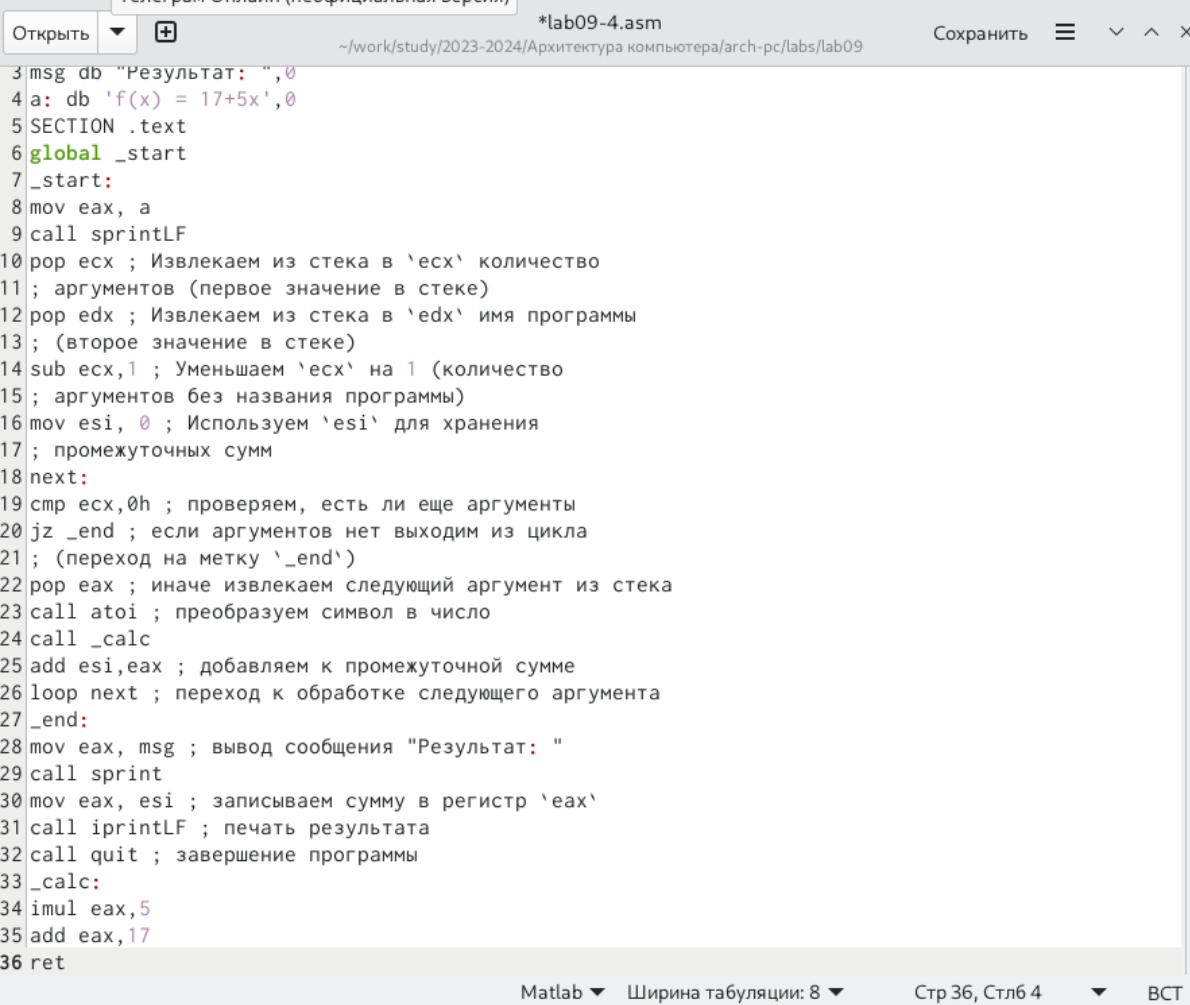
### 3) Выполнение заданий для самостоятельной работы

Копирую файл задания для самостоятельной работы. (рис. 32)

```
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ cp ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08/lab8-4.asm ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-4.asm
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $
```

Рис. 32: Копирование файла

Реализую вычисление значения функции через подпрограмму (рис. 33)



```
*lab09-4.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09

3 msg db "Результат: ",0
4 a: db 'f(x) = 17+5x',0
5 SECTION .text
6 global _start
7 _start:
8 mov eax, a
9 call sprintf
10 pop ecx ; Извлекаем из стека в 'ecx' количество
11 ; аргументов (первое значение в стеке)
12 pop edx ; Извлекаем из стека в 'edx' имя программы
13 ; (второе значение в стеке)
14 sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
15 ; аргументов без названия программы)
16 mov esi, 0 ; Используем 'esi' для хранения
17 ; промежуточных сумм
18 next:
19 cmp ecx,0h ; проверяем, есть ли еще аргументы
20 jz _end ; если аргументов нет выходим из цикла
21 ; (переход на метку '_end')
22 pop eax ; иначе извлекаем следующий аргумент из стека
23 call atoi ; преобразуем символ в число
24 call _calc
25 add esi,eax ; добавляем к промежуточной сумме
26 loop next ; переход к обработке следующего аргумента
27 _end:
28 mov eax, msg ; вывод сообщения "Результат: "
29 call sprint
30 mov eax, esi ; записываем сумму в регистр 'eax'
31 call iprintLF ; печать результата
32 call quit ; завершение программы
33 _calc:
34 imul eax,5
35 add eax,17
36 ret
```

Рис. 33: Копирование файла

Создаю исполняемый файл и убеждаюсь в правильности работы программы

(рис. 34)

```
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ nasm -f elf lab09-4.asm
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ./lab09-4 1
f(x) = 17+5x
Результат: 22
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ./lab09-4 2 5 3
f(x) = 17+5x
Результат: 101
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $
```

Рис. 34: Запуск исполняемого файла

Создаю файл lab09-5.asm (рис. 35)

```
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ touch lab09-5.asm
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $
```

Рис. 35: Создание файла

Вношу в него программу из последнего листинга (рис. 36)

```
Открыть [ ] lab09-5.asm Сохранить [ ] [ ] [ ]
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
```

Рис. 36: Редактирование файла

Создаю исполняемый файл и запускаю его. Вижу, что программа работает неверно и выдает неправильный ответ (правильный ответ:25) (рис. 37)

```
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ nasm -f elf lab09-5.asm
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ./lab09-5
Результат: 10
dsmanjkovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $
```

Рис. 37: Запуск исполняемого файла

Анализирую изменения значений регистров, чтобы выяснить, в чем заключается ошибка (рис. 38)

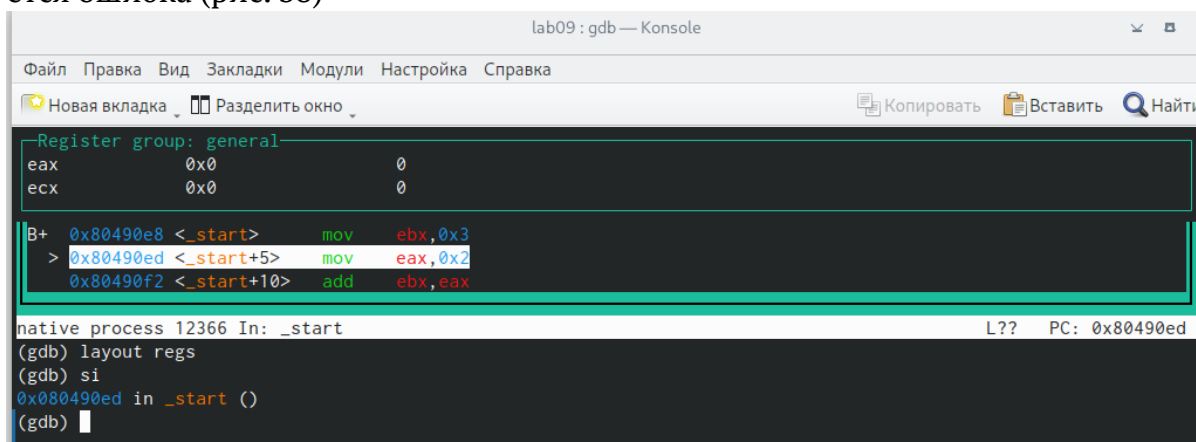


Рис. 38: Выявление ошибки

Нахожу ошибку и исправляю ее в тексте программы (рис. 39)



Рис. 39: Редактирование файла

Создаю исполняемый файл и запускаю его, вижу правильный ответ, программа написана верно (рис. 40)

```

dsmanjovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ nasm -f elf lab09-5.asm
dsmanjovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -c lab09-5 lab09-5.o
ld:lab09-5:1: ignoring invalid character '\177' in script
ld: unrecognized keyword in MRI style script 'ELF'
dsmanjovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
dsmanjovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ./lab09-5
Результат: 25
dsmanjovskaya@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $

```

Рис. 40: Редактирование файла

Листинг 4.1 - Преобразованная программа из лабораторной работы №8.

```

%include 'in_out.asm'

SECTION .data
msg db "Результат:",0
a: db 'f(x) = 17+5x',0

SECTION .text
global _start
_start:
mov eax, a
call sprintLF
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 сумм
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

```

```

call _calc
add esi, eax ; добавляем к промежуточной сумме
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат:"
call sprint
mov eax, esi ; записываем сумму в регистр eax
call iprintLF ; печать результата
call quit ; завершение программы
_calc:
imul eax, 5
add eax, 17
ret

```

Листинг 4.2 - Исправленная программа для вычисления значения выражения

```

#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
```

## **5 Выводы**

При выполнении данной лабораторной работы, я приобрела практический опыт в написании программ с использованием подпрограмм, а также ознакомилась с методами отладки при помощи gdb и его основными возможностями.

## **6 Список литературы**

Архитектура ЭВМ