

# **Отчёт по лабораторной работе №9**

**Дисциплина: Архитектура компьютера**

Бызова Мария Олеговна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>6</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
2.1	Реализация подпрограмм в NASM . . . . .	7
2.2	Отладка программ с помощью GDB . . . . .	9
2.2.1	Добавление точек останова . . . . .	13
2.2.2	Работа с данными программы в GDB . . . . .	14
2.2.3	Обработка аргументов командной строки в GDB . . . . .	17
2.3	Выполнение заданий для самостоятельной работы . . . . .	19
<b>3</b>	<b>Выводы</b>	<b>27</b>

## Список иллюстраций

2.1	Создание необходимой директории и файла . . . . .	7
2.2	Редактирование файла . . . . .	8
2.3	Создание исполняемого файла . . . . .	8
2.4	Запуск исполняемого файла . . . . .	8
2.5	Редактирование файла . . . . .	9
2.6	Создание исполняемого файла . . . . .	9
2.7	Запуск исполняемого файла . . . . .	9
2.8	Создание необходимого файла . . . . .	9
2.9	Редактирование файла . . . . .	10
2.10	Получение исполняемого файла . . . . .	10
2.11	Загрузка исполняемого файла в отладчик gdb . . . . .	10
2.12	Проверка работы программы . . . . .	11
2.13	Запуск программы с брэйкпойнтом . . . . .	11
2.14	Дисассимилированный код программы . . . . .	11
2.15	Синтаксис Intel . . . . .	12
2.16	Режим псевдографики . . . . .	13
2.17	Информация о точках останова . . . . .	13
2.18	Установка точки останова . . . . .	13
2.19	Информация о точках останова . . . . .	14
2.20	Выполняем 5 инструкций командой stepi . . . . .	14
2.21	Информация о регистрах . . . . .	15
2.22	Значение переменной . . . . .	15
2.23	Значение переменной . . . . .	15
2.24	Просмотр инструкции . . . . .	16
2.25	Изменение первого символа переменной msg1 . . . . .	16
2.26	Изменение первого символа переменной msg2 . . . . .	16
2.27	Значение регистра edx . . . . .	16
2.28	Изменение регистра ebx . . . . .	17
2.29	Завершение программы и выход из GDB . . . . .	17
2.30	Копирование файла . . . . .	17
2.31	Создание исполняемого файла . . . . .	18
2.32	Загрузка исполняемого файла в отладчик . . . . .	18
2.33	Установка точки останова и запуск программы . . . . .	18
2.34	Адрес вершины стека . . . . .	18
2.35	Просмотр остальных позиций стека . . . . .	19
2.36	Создание файла . . . . .	19
2.37	Редактирование файла . . . . .	20

2.38	Создание исполняемого файла . . . . .	20
2.39	Запуск исполняемого файла . . . . .	20
2.40	Создание файла . . . . .	20
2.41	Редактирование файла . . . . .	21
2.42	Загрузка файла в отладчик . . . . .	21
2.43	Запуск программы . . . . .	21
2.44	Изменение регистров . . . . .	22
2.45	Редактирование файла . . . . .	22
2.46	Создание исполняемого файла . . . . .	22
2.47	Запуск исполняемого файла . . . . .	23

## Список таблиц

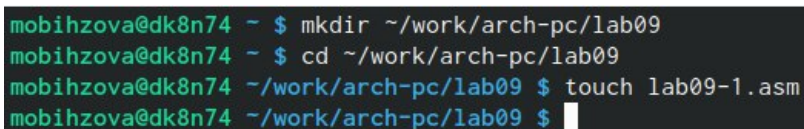
# 1 Цель работы

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

## 2 Выполнение лабораторной работы

### 2.1 Реализация подпрограмм в NASM

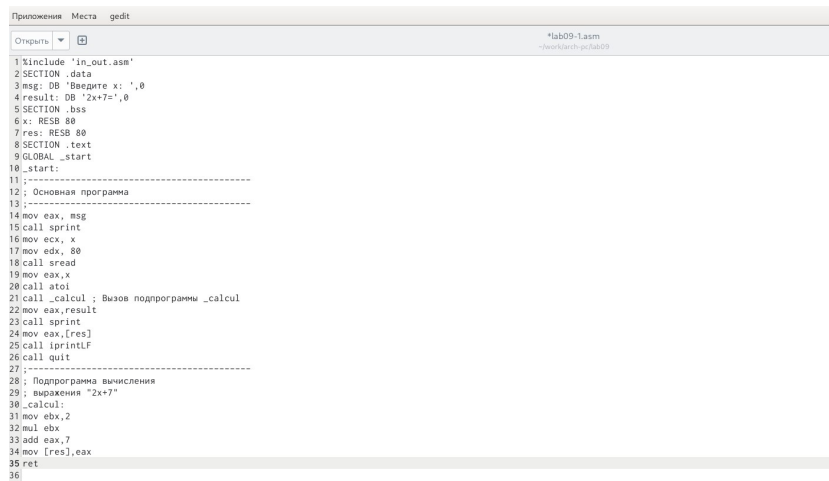
1. С помощью утилиты `mkdir` создаем директорию, в которой будем создавать файлы с программами для лабораторной работы №9. Переходим в созданный каталог с помощью утилиты `cd`. С помощью утилиты `touch` создаем файл `lab09-1.asm` (рис. [2.1]).



```
mobihzova@dk8n74 ~ $ mkdir ~/work/arch-pc/lab09
mobihzova@dk8n74 ~ $ cd ~/work/arch-pc/lab09
mobihzova@dk8n74 ~/work/arch-pc/lab09 $ touch lab09-1.asm
mobihzova@dk8n74 ~/work/arch-pc/lab09 $
```

Рис. 2.1: Создание необходимой директории и файла

2. Внимательно изучив текст программы из листинга 9.1, вводим его в файл `lab09-1.asm` (рис. [2.2]).

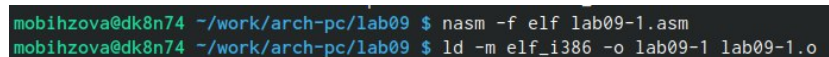


```
Приложения Места gedit
Открыть *lab09-1.asm
~/work/arch-pc/lab09

1#include "in_out.asm"
2SECTION .data
3msg: DB "Введите x: ",0
4result: DB "2x+7=",0
5SECTION .bss
6x: RESB 80
7res: RESB 80
8SECTION .text
9GLOBAL _start
10_start:
11:-----
12: Основная программа
13:-----
14mov eax, msg
15call sprint
16mov ecx, x
17mov edx, 80
18call sread
19mov eax, x
20call atoi
21call _calcul ; Вызов подпрограммы _calcul
22mov eax, result
23call sprint
24mov eax, [res]
25call iprintf
26call quit
27:-----
28: Подпрограмма вычисления
29: выражения "2x+7"
30_calcul:
31mov ebx, 2
32mul ebx
33add eax, 7
34mov [res], eax
35ret
36
```

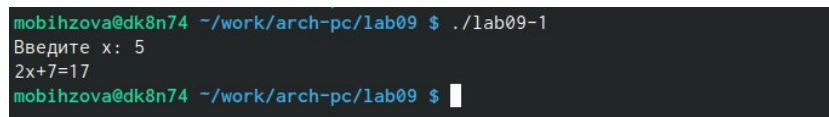
Рис. 2.2: Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [2.3], [2.4]).



```
mobihzova@dk8n74 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
mobihzova@dk8n74 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
```

Рис. 2.3: Создание исполняемого файла



```
mobihzova@dk8n74 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 5
2x+7=17
mobihzova@dk8n74 ~/work/arch-pc/lab09 $
```

Рис. 2.4: Запуск исполняемого файла

Изменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$  (рис. [2.5]).



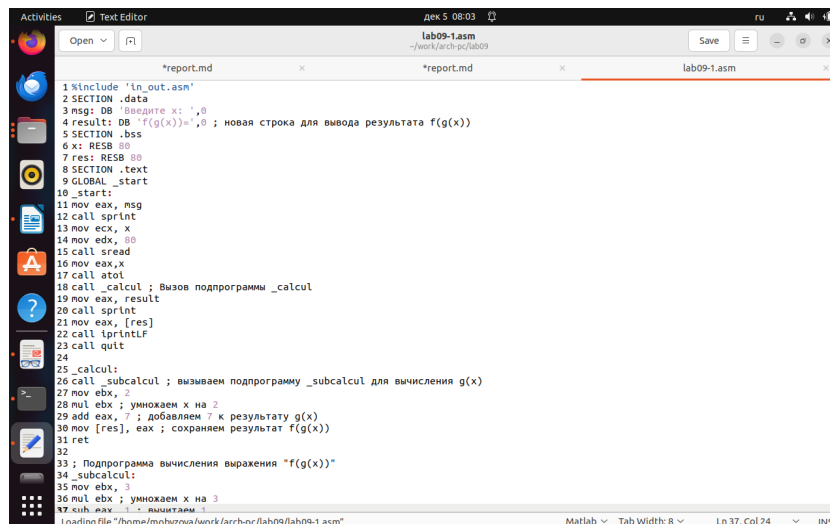


Рис. 2.5: Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [2.6], [2.7]).

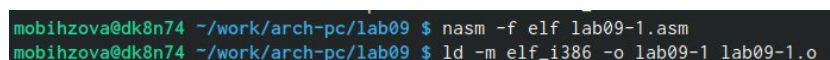


Рис. 2.6: Создание исполняемого файла

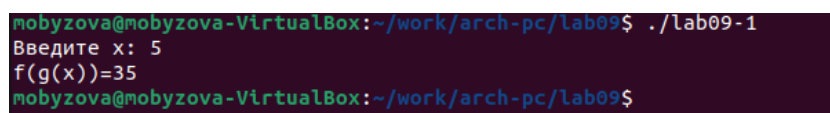


Рис. 2.7: Запуск исполняемого файла

## 2.2 Отладка программ с помощью GDB

Создаем новый файл lab09-2.asm в каталоге (рис. [2.8]).

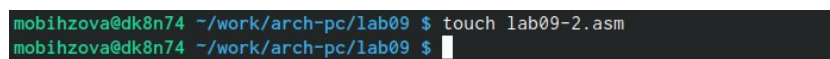


Рис. 2.8: Создание необходимого файла

Открываем файл и заполняем его в соответствии с листингом 9.2 (рис. [2.9]).

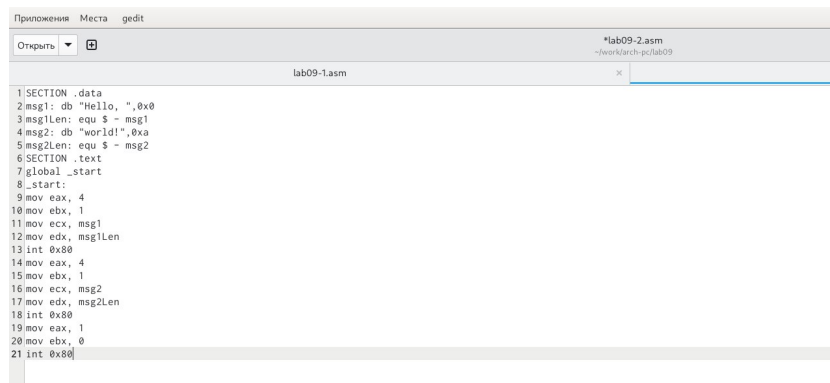


Рис. 2.9: Редактирование файла

Получим исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’ (рис. [2.10]).

```
mobihzova@dk8n74 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
mobihzova@dk8n74 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
mobihzova@dk8n74 ~/work/arch-pc/lab09 $
```

Рис. 2.10: Получение исполняемого файла

Загрузим исполняемый файл в отладчик gdb (рис. [2.11]).

```
mobihzova@dk8n74 ~/work/arch-pc/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. 2.11: Загрузка исполняемого файла в отладчик gdb

Проверим работу программы, запустив ее в оболочке GDB с помощью команды `run` (рис. [2.12]).

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/o/mobihzova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 7485) exited normally]
(gdb)
```

Рис. 2.12: Проверка работы программы

Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её (рис. [2.13]).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/o/mobihzova/work/arch-pc/lab09/lab09-2

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

Рис. 2.13: Запуск программы с брэйкпоинтом

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. [2.14]).

```
9      mov eax, 4
(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. 2.14: Дисассимилированный код программы

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. [2.15]).

```

(gdb) set disassembly-flavor intel
(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. 2.15: Синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel:

1. Порядок операндов: В АТТ синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный - вторым.
2. Разделители: В АТТ синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).
3. Префиксы размера операндов: В АТТ синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как "b" (byte), "w" (word), "l" (long) и "q" (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как "b", "w", "d" и "q".
4. Знак операндов: В АТТ синтаксисе операнды с позитивными значениями предваряются символом `".Intel"`.
5. Обозначение адресов: В АТТ синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.
6. Обозначение регистров: В АТТ синтаксисе обозначение регистра начинается с символа "%". В Intel синтаксисе обозначение регистра может начинаться с символа "R" или "E" (например, `%eax` или `RAX`).

Включим режим псевдографики для более удобного анализа программы (рис. [2.16]).

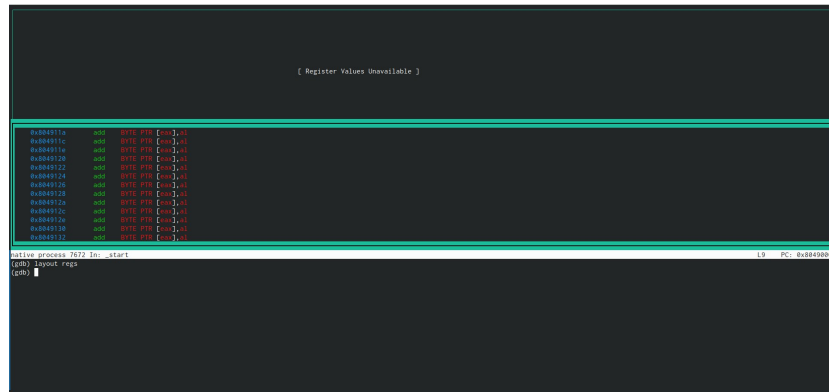


Рис. 2.16: Режим псевдографики

### 2.2.1 Добавление точек останова

Проверим установленные точки останова с помощью команды `info breakpoints` (рис. [2.17]).

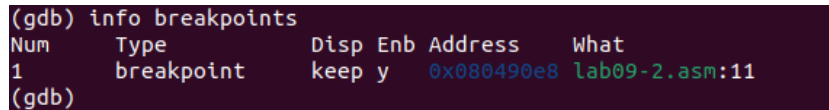


Рис. 2.17: Информация о точках останова

Определим адрес предпоследней инструкции (`mov ebx,0x0`) и установим точку останова (рис. [2.18]).

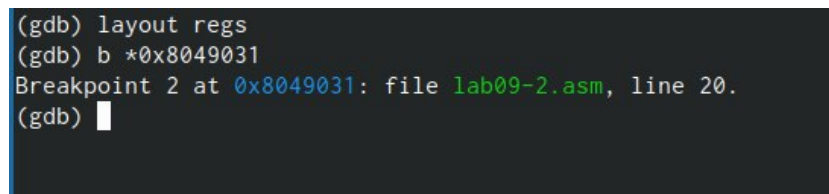


Рис. 2.18: Установка точки останова

Посмотрим информацию о всех установленных точках останова (рис. [2.19]).

```
(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. 2.19: Информация о точках останова

## 2.2.2 Работа с данными программы в GDB

Выполните 5 инструкций с помощью команды `stepi` (или `si`) (рис. [2.20]).

```
(gdb) run
Starting program: /home/mobyzova/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb) stepi
10     mov ebx, 1
(gdb) stepi
11     mov ecx, msg1
(gdb) stepi
12     mov edx, msg1Len
(gdb) stepi
13     int 0x80
(gdb) stepi
Hello, 14     mov eax, 4
(gdb)
```

Рис. 2.20: Выполняем 5 инструкций командой `stepi`

Посмотрев содержимое регистров с помощью команды `info registers` (или `i r`), замечаем, что во время выполнения команд менялись регистры: `ebx`, `ecx`, `edx`, `eax`, `ebp` (рис. [2.21]).

```

(gdb) info registers
eax            0x8                8
ecx            0x804a000          134520832
edx            0x8                8
ebx            0x1                1
esp            0xffffd0f0         0xffffd0f0
ebp            0x0                0x0
esi            0x0                0
edi            0x0                0
eip            0x8049016          0x8049016 <_start+22>
eflags         0x202             [ IF ]
cs             0x23              35
ss             0x2b              43
ds             0x2b              43
es             0x2b              43
fs             0x0                0
gs             0x0                0
(gdb)

```

Рис. 2.21: Информация о регистрах

С помощью команды `x &` также можно посмотреть содержимое переменной. Посмотрим значение переменной `msg1` по имени (рис. [2.22]).

```

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

```

Рис. 2.22: Значение переменной

Смотрим значение переменной `msg2` по адресу (рис. [2.23]).

```

(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 2.23: Значение переменной

Посмотрим инструкцию `mov esx, msg2` которая записывает в регистр `esx` адрес переменной `msg2` (рис. [2.24]).

```
(gdb) x/1sb 0x8049020
0x8049020 <_start+32>: "\271\b\240\004\b\272\a"
(gdb)
```

Рис. 2.24: Просмотр инструкции

Изменим первый символ переменной msg1 (рис. [2.25]).

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

Рис. 2.25: Изменение первого символа переменной msg1

Изменим первый символ переменной msg2 (рис. [2.26]).

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

Рис. 2.26: Изменение первого символа переменной msg2

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. [2.27]).

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

Рис. 2.27: Значение регистра edx

Изменим регистр ebx (рис. [2.28]).



```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$5 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
(gdb) █
```

Рис. 2.28: Изменение регистра ebx

Выводятся разные значения, так как именно команда без кавычек присваивает регистру вводимое значение.

Завершим выполнение программы с помощью команды `continue` (сокращенно `c`) и выйдем из GDB с помощью команды `quit` (сокращенно `q`) (рис. [2.29]).

```
(gdb) continue
Continuing.
hello, World!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb) quit █
```

Рис. 2.29: Завершение программы и выход из GDB

### 2.2.3 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 (Листинг 8.2) в файл с именем `lab09-3.asm` (рис. [2.30]).

```
mobyzova@mobyzova-VirtualBox: $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
mobyzova@mobyzova-VirtualBox: $ █
```

Рис. 2.30: Копирование файла

Далее создадим исполняемый файл (рис. [2.31]).

```
mobyzoa@mobyzoa-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
mobyzoa@mobyzoa-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
mobyzoa@mobyzoa-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.31: Создание исполняемого файла

Загрузим исполняемый файл в отладчик, указав аргументы (рис. [2.32]).

```
mobyzoa@mobyzoa-VirtualBox:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 "аргумент 3"
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 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-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...
(gdb)
```

Рис. 2.32: Загрузка исполняемого файла в отладчик

Для начала установим точку останова перед первой инструкцией в программе и запустим ее (рис. [2.33]).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/mobyzoa/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

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

Рис. 2.33: Установка точки останова и запуск программы

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы) (рис. [2.34]).

```
(gdb) x/x $esp
0xfffff0b0: 0x00000005
(gdb)
```

Рис. 2.34: Адрес вершины стека

Посмотрим остальные позиции стека (рис. [2.35]).

```

(gdb) x/s *(void**)($esp + 4)
0x1ffffd282: "/home/mobyzova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0x1ffffd2ac: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0x1ffffd2be: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0x1ffffd2cf: "2"
(gdb) x/s *(void**)($esp + 20)
0x1ffffd2d1: "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x1: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 2.35: Просмотр остальных позиций стека

Шаг изменения адреса равен 4, потому что в большинстве архитектур процессоров размер слова (или размер указателя) составляет 4 байта. Это означает, что каждый раз, когда мы обращаемся к следующему элементу в стеке, мы увеличиваем адрес на 4, чтобы перейти к следующему слову данных. Таким образом, шаг изменения адреса равен 4 для обеспечения корректного доступа к данным в стеке.

## 2.3 Выполнение заданий для самостоятельной работы

1. Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму.

Создаем файл для выполнения задания (рис. [2.36]).

```

mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab09$ touch lab09-4.asm

```

Рис. 2.36: Создание файла

Вставляем отредактированную программу из лабораторной работы №8 с добавлением подпрограммы (рис. [2.37]).

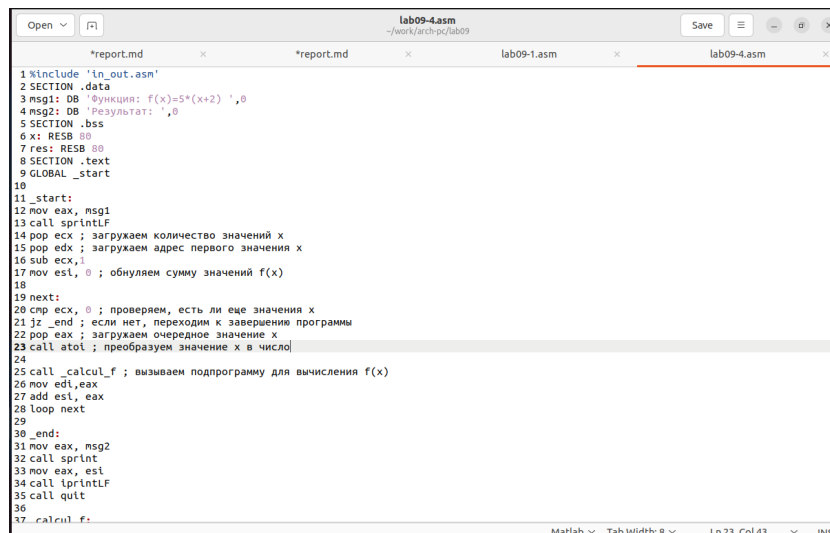


Рис. 2.37: Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [2.38], [2.39]).

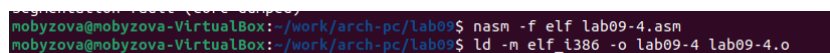


Рис. 2.38: Создание исполняемого файла

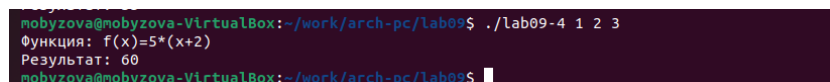


Рис. 2.39: Запуск исполняемого файла

- В листинге 9.3 приведена программа вычисления выражения  $(3 + 2) * 4 + 5$ . При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

Создаем файл для выполнения задания (рис. [2.40]).

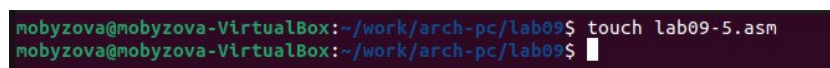


Рис. 2.40: Создание файла

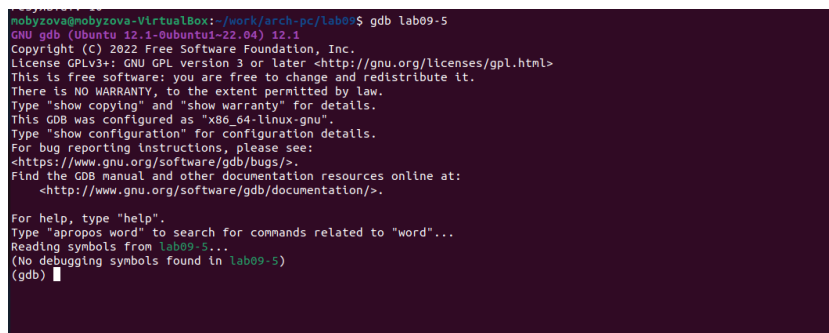
Вставляем программу (рис. [2.41]).



```
1 %include 'in_out.asm'
2 SECTION .data
3 dlv: 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 fprintf
20 call quit
```

Рис. 2.41: Редактирование файла

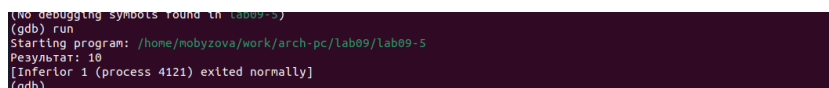
Создаем исполняемый файл и запускаем его в отладчике GDB. Смотрим на изменение регистров по ходу программы (рис. [2.42], [2.43], [2.44]).



```
nobyzo@nobyzo-VirtualBox:~/work/arch-pc/lab09$ gdb lab09-5
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 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-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 lab09-5...
(no debugging symbols found in lab09-5)
(gdb)
```

Рис. 2.42: Загрузка файла в отладчик



```
(no debugging symbols found in lab09-5)
(gdb) run
Starting program: /home/nobyzo/work/arch-pc/lab09/lab09-5
Результат: 10
[Inferior 1 (process 4121) exited normally]
(gdb)
```

Рис. 2.43: Запуск программы

```
(gdb) break *0x080490fb
Breakpoint 3 at 0x080490fb
(gdb) s
Single stepping until exit from function _start,
which has no line number information.

Breakpoint 3, 0x080490fb in _start ()
(gdb) info registers
eax            0x0            0
ecx            0x4            4
edx            0x0            0
ebx            0x5            5
esp            0xffffd0f0     0xffffd0f0
ebp            0x0            0x0
esi            0x0            0
edi            0x0            0
eip            0x080490fb     0x080490fb <_start+19>
eflags         0x202         [ IF ]
cs             0x23          35
ss             0x2b          43
ds             0x2b          43
es             0x2b          43
fs             0x0            0
gs             0x0            0
(gdb)
```

Рис. 2.44: Изменение регистров

Обнаружив ошибку неправильной записи регистров, корректируем программу (рис. [2.45]).



```
lab09-5.asm
~/work/arch-pc/lab09
Save

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 ecx,2
10 add eax,ebx
11 mov ecx,4
12 mul ecx
13 add eax,ecx
14 mov edi,eax
15 ; ----
16 mov eax,div
17 call sprintf
18 mov eax,edi
19 call fprintf
20 call quit
```

Рис. 2.45: Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [2.46], [2.47]).

```
mobyzova@mobyzova-VirtualBox: /work/arch-pc/lab09$ nasm -f elf lab09-5.asm
mobyzova@mobyzova-VirtualBox: /work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
mobyzova@mobyzova-VirtualBox: /work/arch-pc/lab09$
```

Рис. 2.46: Создание исполняемого файла

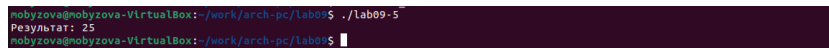


Рис. 2.47: Запуск исполняемого файла

### Листинг №1. Изменённая программа

```
%include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x))=',0 ; новая строка для вывода результата f(g(x))

SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start
_start:
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
```

```

_calcul:
call _subcalcul ; вызываем подпрограмму _subcalcul для вычисления g(x)
mov ebx, 2
mul ebx ; умножаем x на 2
add eax, 7 ; добавляем 7 к результату g(x)
mov [res], eax ; сохраняем результат f(g(x))
ret

```

; Подпрограмма вычисления выражения "f(g(x))"

```

_subcalcul:
mov ebx, 3
mul ebx ; умножаем x на 3
sub eax, 1 ; вычитаем 1
ret

```

## Листинг №2. Задание для самостоятельной работы №1

```

#include 'in_out.asm'

SECTION .data
msg1: DB 'Функция: f(x)=5*(x+2) ',0
msg2: DB 'Результат: ',0

SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start

_start:
mov eax, msg1
call sprintf
pop ecx ; загружаем количество значений x

```



```

pop edx ; загружаем адрес первого значения x
sub ecx,1
mov esi, 0 ; обнуляем сумму значений f(x)

next:
cmp ecx, 0 ; проверяем, есть ли еще значения x
jz _end ; если нет, переходим к завершению программы
pop eax ; загружаем очередное значение x
call atoi ; преобразуем значение x в число

call _calcul_f ; вызываем подпрограмму для вычисления f(x)
mov edi, eax
add esi, eax
loop next

_end:
mov eax, msg2
call sprint
mov eax, esi
call iprintLF
call quit

_calcul_f:
add eax, 2 ; прибавляем 2 к x
mov ebx, 5
mul ebx ; умножаем результат на 5
ret

```

### Листинг №3. Задание для самостоятельной работы №2

```
%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 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

```

## 3 Выводы

В ходе выполнения лабораторной работы мы приобрели навыки написания программ с использованием подпрограмм, познакомились с методами отладки при помощи GDB и его основными возможностями.