

# **Отчёта по лабораторной работе 10**

**Понятие подпрограммы. Отладчик GDB.**

Дев Авинаш НКАбд-05-22

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	27

## Список иллюстраций

2.1	Файл lab10-1.asm . . . . .	7
2.2	Работа программы lab10-1.asm . . . . .	8
2.3	Файл lab10-1.asm . . . . .	9
2.4	Работа программы lab10-1.asm . . . . .	9
2.5	Файл lab10-2.asm . . . . .	10
2.6	Работа программы lab10-2.asm в отладчике . . . . .	11
2.7	дисассимилированный код . . . . .	12
2.8	дисассимилированный код в режиме интел . . . . .	13
2.9	точка остановки . . . . .	14
2.10	изменение регистров . . . . .	15
2.11	изменение регистров . . . . .	16
2.12	изменение значения переменной . . . . .	17
2.13	вывод значения регистра . . . . .	18
2.14	вывод значения регистра . . . . .	19
2.15	вывод значения регистра . . . . .	20
2.16	Файл lab10-4.asm . . . . .	21
2.17	Работа программы lab10-4.asm . . . . .	22
2.18	код с ошибкой . . . . .	23
2.19	отладка . . . . .	24
2.20	код исправлен . . . . .	25
2.21	проверка работы . . . . .	26

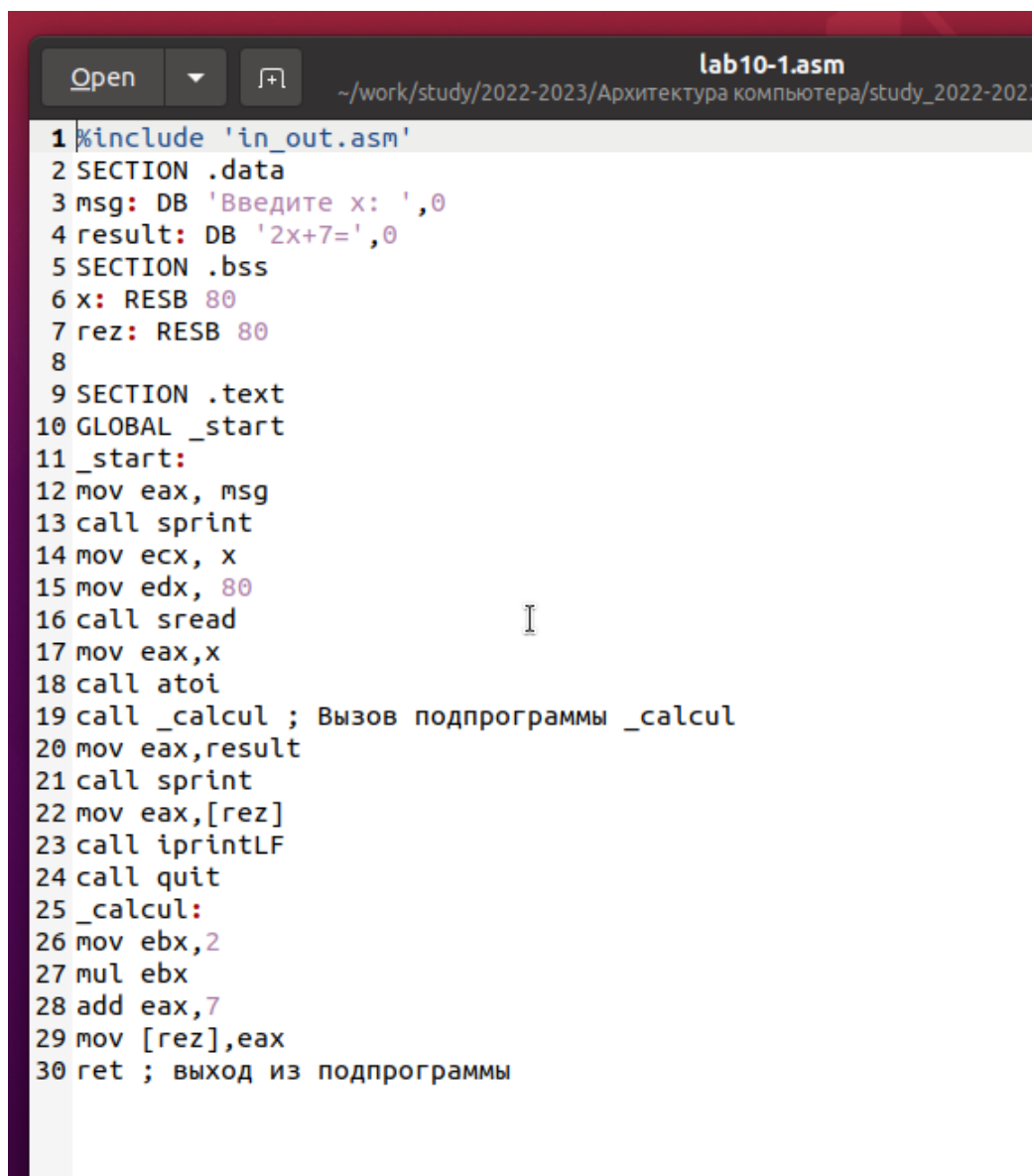
## Список таблиц

# 1 Цель работы

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

## 2 Выполнение лабораторной работы

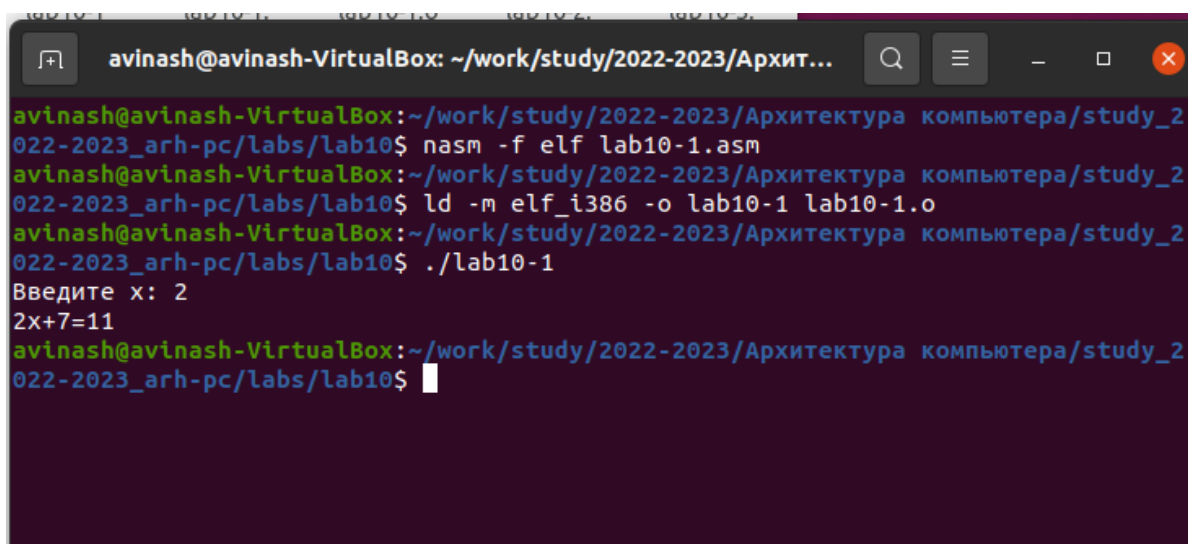
1. Создайте каталог для выполнения лабораторной работы № 10, перейдите в него и создайте файл lab10-1.asm:
2. В качестве примера рассмотрим программу вычисления арифметического выражения  $f(x) = 2x+7$  с помощью подпрограммы calcul. В данном примере  $x$  вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы (Листинг 10.1). (рис. 2.1, 2.2)



```
lab10-1.asm
~/work/study/2022-2023/Архитектура компьютера/study_2022-2023/

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

Рис. 2.1: Файл lab10-1.asm

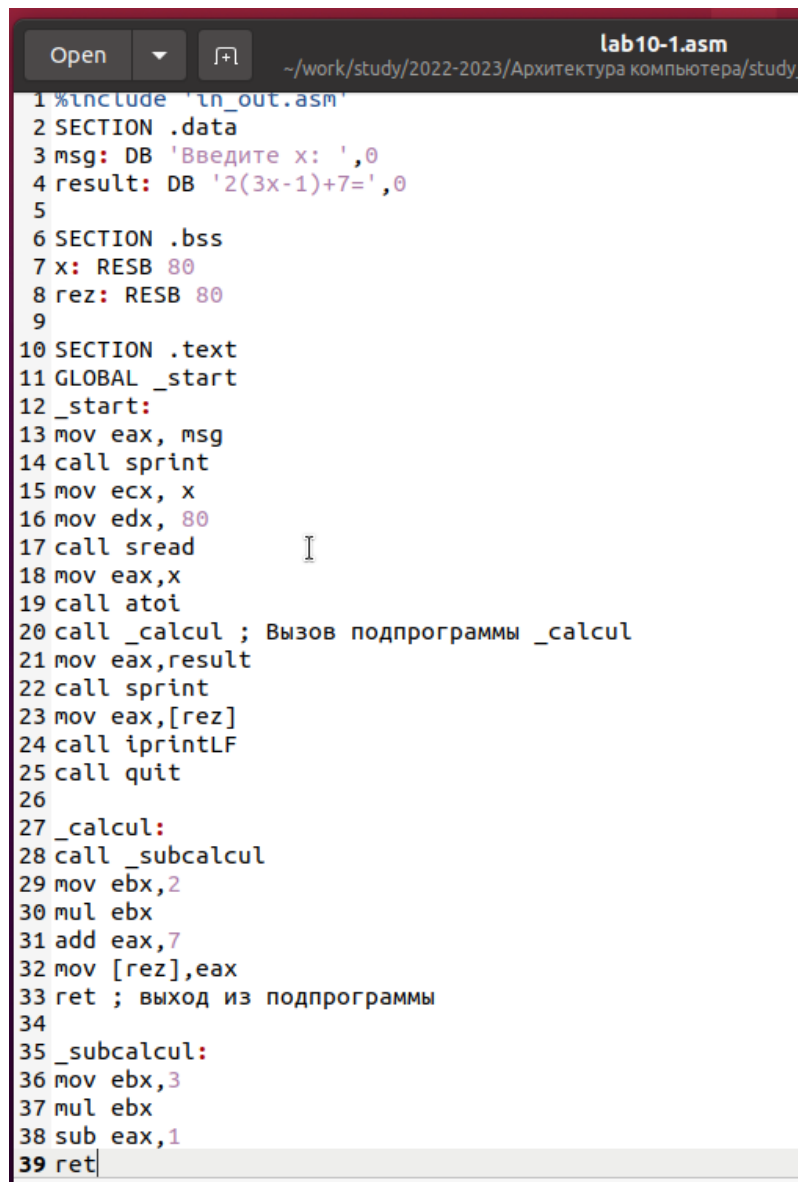


```
avinash@avinash-VirtualBox: ~/work/study/2022-2023/Архитектура компьютера/study_2022-2023_arh-pc/labs/lab10$ nasm -f elf lab10-1.asm
avinash@avinash-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/study_2022-2023_arh-pc/labs/lab10$ ld -m elf_i386 -o lab10-1 lab10-1.o
avinash@avinash-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/study_2022-2023_arh-pc/labs/lab10$ ./lab10-1
Введите x: 2
2x+7=11
avinash@avinash-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/study_2022-2023_arh-pc/labs/lab10$
```

Рис. 2.2: Работа программы lab10-1.asm

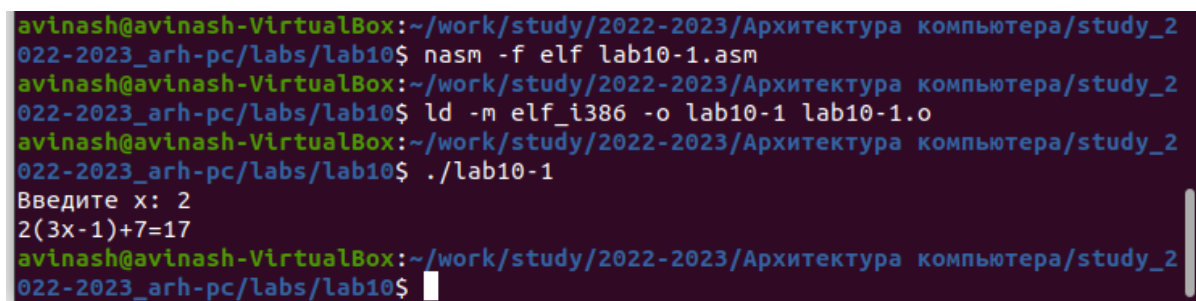
3. Измените текст программы, добавив подпрограмму subcalcul в подпрограмму calcul, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$  (рис. 2.3, 2.4)





```
lab10-1.asm
~/work/study/2022-2023/Архитектура компьютера/study
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2(3x-1)+7=',0
5
6 SECTION .bss
7 x: RESB 80
8 rez: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13 mov eax, msg
14 call sprint
15 mov ecx, x
16 mov edx, 80
17 call sread
18 mov eax, x
19 call atoi
20 call _calcul ; Вызов подпрограммы _calcul
21 mov eax, result
22 call sprint
23 mov eax, [rez]
24 call iprintLF
25 call quit
26
27 _calcul:
28 call _subcalcul
29 mov ebx, 2
30 mul ebx
31 add eax, 7
32 mov [rez], eax
33 ret ; выход из подпрограммы
34
35 _subcalcul:
36 mov ebx, 3
37 mul ebx
38 sub eax, 1
39 ret
```

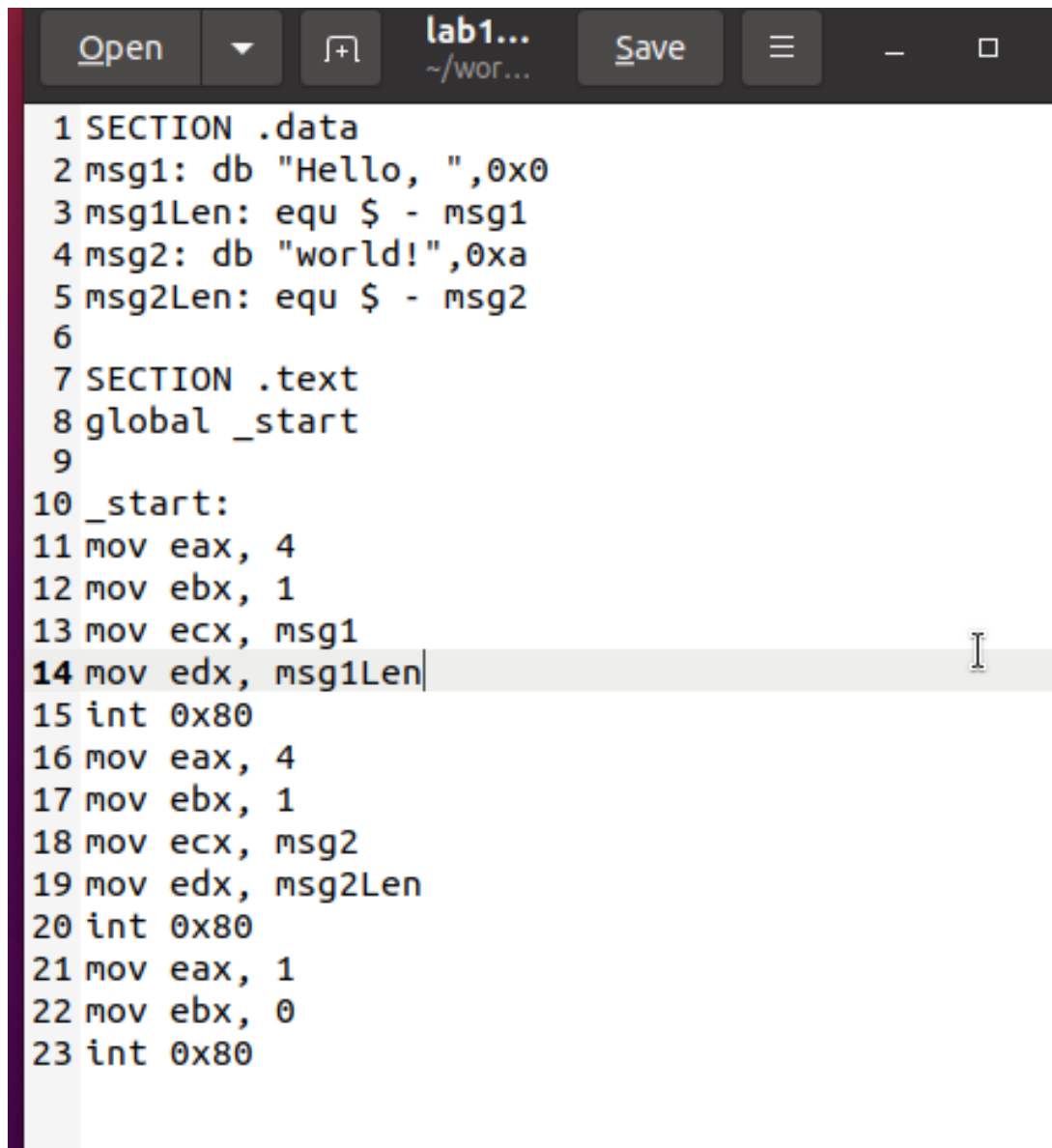
Рис. 2.3: Файл lab10-1.asm



```
avinash@avinash-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/study_2
022-2023_arh-pc/labs/lab10$ nasm -f elf lab10-1.asm
avinash@avinash-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/study_2
022-2023_arh-pc/labs/lab10$ ld -m elf_i386 -o lab10-1 lab10-1.o
avinash@avinash-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/study_2
022-2023_arh-pc/labs/lab10$ ./lab10-1
Введите x: 2
2(3x-1)+7=17
avinash@avinash-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/study_2
022-2023_arh-pc/labs/lab10$
```

Рис. 2.4: Работа программы lab10-1.asm

4. Создайте файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!): (рис. 2.5)

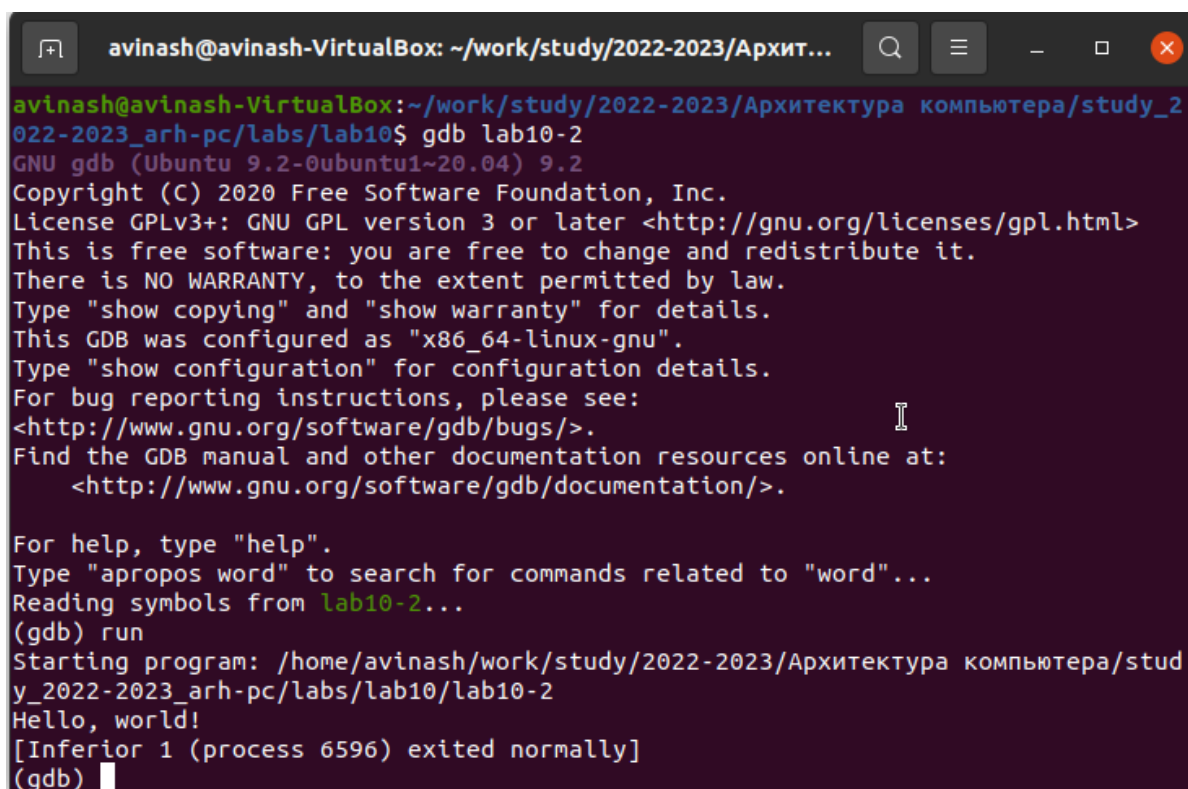


```
1 SECTION .data
2 msg1: db "Hello, ",0x0
3 msg1Len: equ $ - msg1
4 msg2: db "world!",0xa
5 msg2Len: equ $ - msg2
6
7 SECTION .text
8 global _start
9
10 _start:
11 mov eax, 4
12 mov ebx, 1
13 mov ecx, msg1
14 mov edx, msg1Len
15 int 0x80
16 mov eax, 4
17 mov ebx, 1
18 mov ecx, msg2
19 mov edx, msg2Len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

Рис. 2.5: Файл lab10-2.asm

Получите исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. Загрузите исполняемый файл в отладчик gdb: Проверьте работу программы, запустив ее в оболочке GDB с помощью ко-

манды run (сокращённо r):(рис. 2.6)

A screenshot of a terminal window titled 'avinash@avinash-VirtualBox: ~/work/study/2022-2023/Архит...'. The terminal shows the user running 'gdb lab10-2'. It displays the GNU GDB version 9.2, copyright information, and license details. The user then enters 'run', and the program 'lab10-2' starts, printing 'Hello, world!'. Finally, the program exits normally, and the user returns to the GDB prompt.

```
avinash@avinash-VirtualBox: ~/work/study/2022-2023/Архит...
avinash@avinash-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/study_2
022-2023_arh-pc/labs/lab10$ gdb lab10-2
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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:
<http://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 lab10-2...
(gdb) run
Starting program: /home/avinash/work/study/2022-2023/Архитектура компьютера/stud
y_2022-2023_arh-pc/labs/lab10/lab10-2
Hello, world!
[Inferior 1 (process 6596) exited normally]
(gdb)
```

Рис. 2.6: Работа программы lab10-2.asm в отладчике

Для более подробного анализа программы установите брейкпоинт на метку start, с которой начинается выполнение любой ассемблерной программы, и запустите её. Посмотрите дисассимилированный код программы (рис. 2.7, 2.8)

```
avinash@avinash-VirtualBox: ~/work/study/2022-2023/Архит...
(gdb) break _start
Breakpoint 1 at 0x8049000
(gdb) run
Starting program: /home/avinash/work/study/2022-2023/Архитектура компьютера/study_2022-2023_arh-pc/labs/lab10/lab10-2

Breakpoint 1, 0x8049000 in _start ()
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x8049000 <+0>:      mov     $0x4,%eax
      0x8049005 <+5>:      mov     $0x1,%ebx
      0x804900a <+10>:     mov     $0x804a000,%ecx
      0x804900f <+15>:     mov     $0x8,%edx
      0x8049014 <+20>:     int     $0x80
      0x8049016 <+22>:     mov     $0x4,%eax
      0x804901b <+27>:     mov     $0x1,%ebx
      0x8049020 <+32>:     mov     $0x804a008,%ecx
      0x8049025 <+37>:     mov     $0x7,%edx
      0x804902a <+42>:     int     $0x80
      0x804902c <+44>:     mov     $0x1,%eax
      0x8049031 <+49>:     mov     $0x0,%ebx
      0x8049036 <+54>:     int     $0x80
End of assembler dump.
(gdb) 
```

Рис. 2.7: дисассимилированный код

```
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) 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. 2.8: дисассимилированный код в режиме интел

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверьте 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. Определите адрес предпоследней инструкции (`mov ebx,0x0`) и установите точку.(рис. 2.9)

The screenshot shows a GDB debugger window with the title bar 'avinash@avinash-VirtualBox: ~/work/study/2022-2023/Архитектура комп...'. The window is divided into several sections. At the top, the 'Register group: general' is displayed, showing the following values:

Register	Value
eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xffffd0e0
ebp	0x0

Below the register list, a list of assembly instructions is shown, with the first instruction highlighted in blue:

```
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
```

At the bottom of the window, the status bar shows 'native process 6600 In: \_start' and 'PC: 0x8049000'. The command prompt shows '(gdb) layout regs' and '(gdb)'.

Рис. 2.9: точка остановки

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров. (рис. 2.11 2.12)

```
avinish@avinash-VirtualBox: ~/work/study/2022-2023/Архитектура комп...
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd0e0 0xffffd0e0
ebp      0x0      0x0

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

native process 6600 In: _start L?? PC: 0x8049000
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) █
```

Рис. 2.10: изменение регистров

```
avinash@avinash-VirtualBox: ~/work/study/2022-2023/Архитектура комп...

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0e0 0xffffd0e0
ebp      0x0      0x0

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

native process 6600 In: _start
ds      0x2b      43
es      0x2b      43
fs      0x0      0
gs      0x0      0
(gdb)
(gdb) si0x08049005 in _start ()
(gdb)
(gdb) si0x0804900a in _start ()
(gdb)
(gdb) si0x0804900f in _start ()
(gdb) si
0x08049014 in _start ()
(gdb)
```

Рис. 2.11: изменение регистров

Посмотрите значение переменной msg1 по имени Посмотрите значение переменной msg2 по адресу Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. Измените первый символ переменной msg1 Замените любой символ во второй переменной msg2. (рис. 2.12)



```
avinash@avinash-VirtualBox: ~/work/study/2022-2023/Архитектура комп...
Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0e0 0xffffd0e0
ebp      0x0      0x0

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

native process 6600 In: _start L?? PC: 0x8049014
(gdb) si
0x08049014 in _start ()
(gdb)
(gdb) x/1sb &msg10x804a000 <msg1>: "Hello, "
(gdb)
(gdb) x/1sb 0x804a0080x804a008 <msg2>: "world!\n"
(gdb)
(gdb)
(gdb) x/1sb &msg10x804a000 <msg1>: "hello, "
(gdb)
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "Lorld!\n"
(gdb)
```

Рис. 2.12: изменение значения переменной

Выведете в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. С помощью команды set измените значение регистра ebx:(рис. 2.13)

```
avinash@avinash-VirtualBox: ~/work/study/2022-2023/Архитектура комп...
Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0e0 0xffffd0e0
ebp      0x0      0x0

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

native process 6600 In: _start L?? PC: 0x8049014
(gdb)
(gdb) p/t $eax$2 = 100
(gdb)
(gdb) p/s $ecx$3 = 134520832
(gdb)
(gdb) p/x $ecx$4 = 0x804a000
(gdb)
(gdb) p/s $edx$5 = 8
(gdb)
(gdb) p/t $edx$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb)
```

Рис. 2.13: вывод значения регистра

С помощью команды set измените значение регистра ebx:(рис. 2.14)

```
Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd0e0 0xffffd0e0
ebp      0x0      0x0

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

native process 6600 In: _start L?? PC: 0x8049014
(gdb)
(gdb) p/s $edx$5 = 8
(gdb)
(gdb) p/t $edx$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb)
(gdb)
(gdb) p/s $ebx$8 = 50
(gdb)
(gdb) p/s $ebx
$9 = 2
(gdb)
```

Рис. 2.14: вывод значения регистра

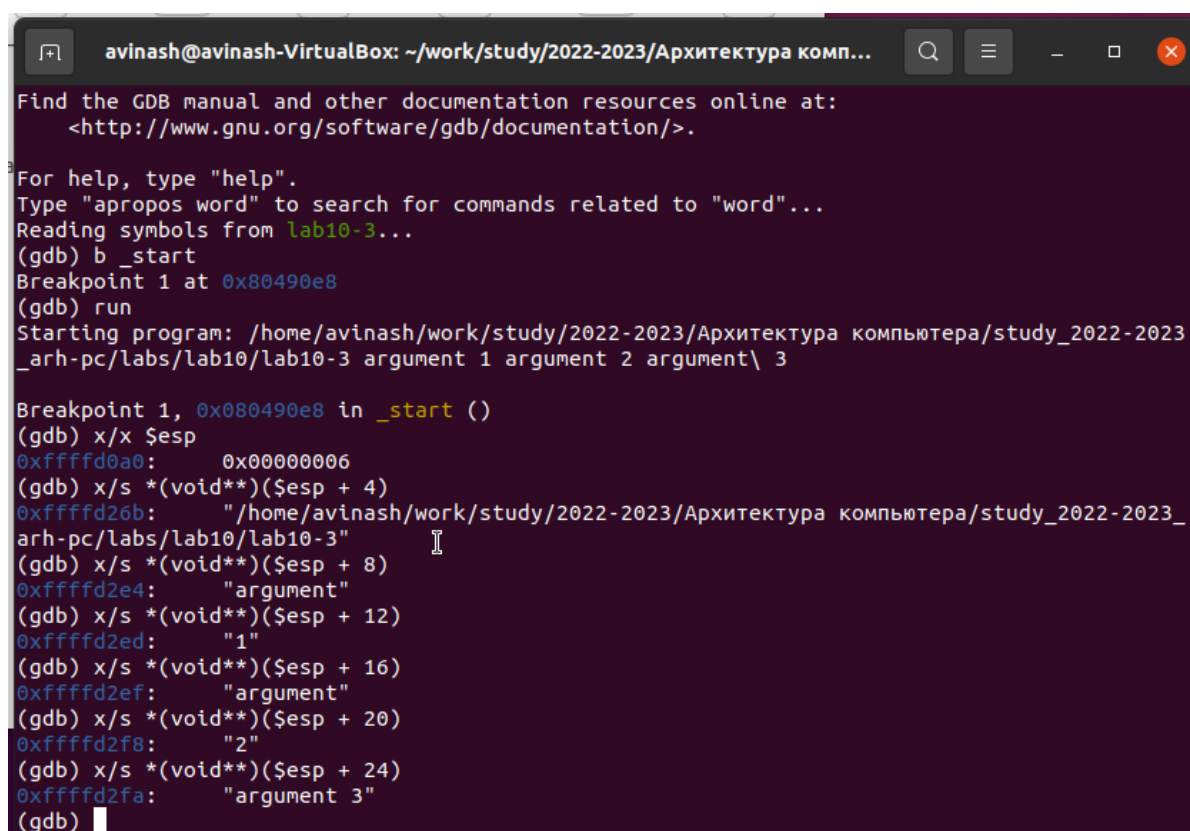
5. Скопируйте файл lab9-2.asm, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки. Создайте исполняемый файл. Для загрузки в gdb программы с аргументами необходимо использовать ключ `-args`. Загрузите исполняемый файл в отладчик, указав аргументы

Для начала установим точку останова перед первой инструкцией в программе и запустим ее.

Адрес вершины стека храниться в регистре `esp` и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы): Как видно, число аргументов равно 5 – это имя программы lab10-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Посмотрите остальные позиции стека – по адресу `[esp+4]` располагается адрес в памяти где находится имя программы, по адресу `[esp+8]` храниться адрес первого

аргумента, по адресу [esp+12] – второго и т.д. (рис. 2.15)



```
avinash@avinash-VirtualBox: ~/work/study/2022-2023/Архитектура комп...
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

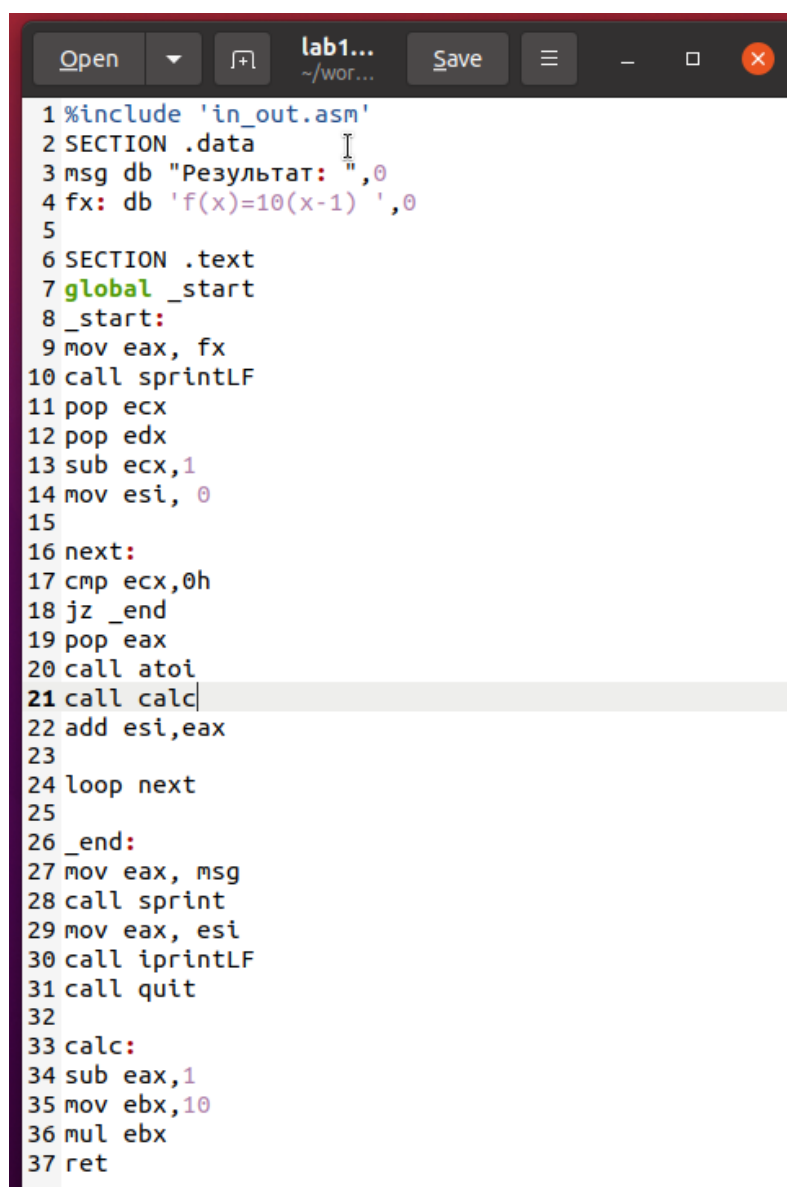
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab10-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8
(gdb) run
Starting program: /home/avinash/work/study/2022-2023/Архитектура компьютера/study_2022-2023_
_arh-pc/labs/lab10/lab10-3 argument 1 argument 2 argument\ 3

Breakpoint 1, 0x080490e8 in _start ()
(gdb) x/x $esp
0xffffd0a0: 0x00000006
(gdb) x/s *(void**)(esp + 4)
0xffffd26b: "/home/avinash/work/study/2022-2023/Архитектура компьютера/study_2022-2023_
_arh-pc/labs/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd2e4: "argument"
(gdb) x/s *(void**)(esp + 12)
0xffffd2ed: "1"
(gdb) x/s *(void**)(esp + 16)
0xffffd2ef: "argument"
(gdb) x/s *(void**)(esp + 20)
0xffffd2f8: "2"
(gdb) x/s *(void**)(esp + 24)
0xffffd2fa: "argument 3"
(gdb)
```

Рис. 2.15: вывод значения регистра

Объясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] - шаг равен размеру переменной - 4 байтам.

- Преобразуйте программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму. (рис. 2.16 2.17)



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)=10(x-1) ',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintf
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 call calc
22 add esi,eax
23
24 loop next
25
26 _end:
27 mov eax, msg
28 call sprintf
29 mov eax, esi
30 call iprintLF
31 call quit
32
33 calc:
34 sub eax,1
35 mov ebx,10
36 mul ebx
37 ret
```

Рис. 2.16: Файл lab10-4.asm

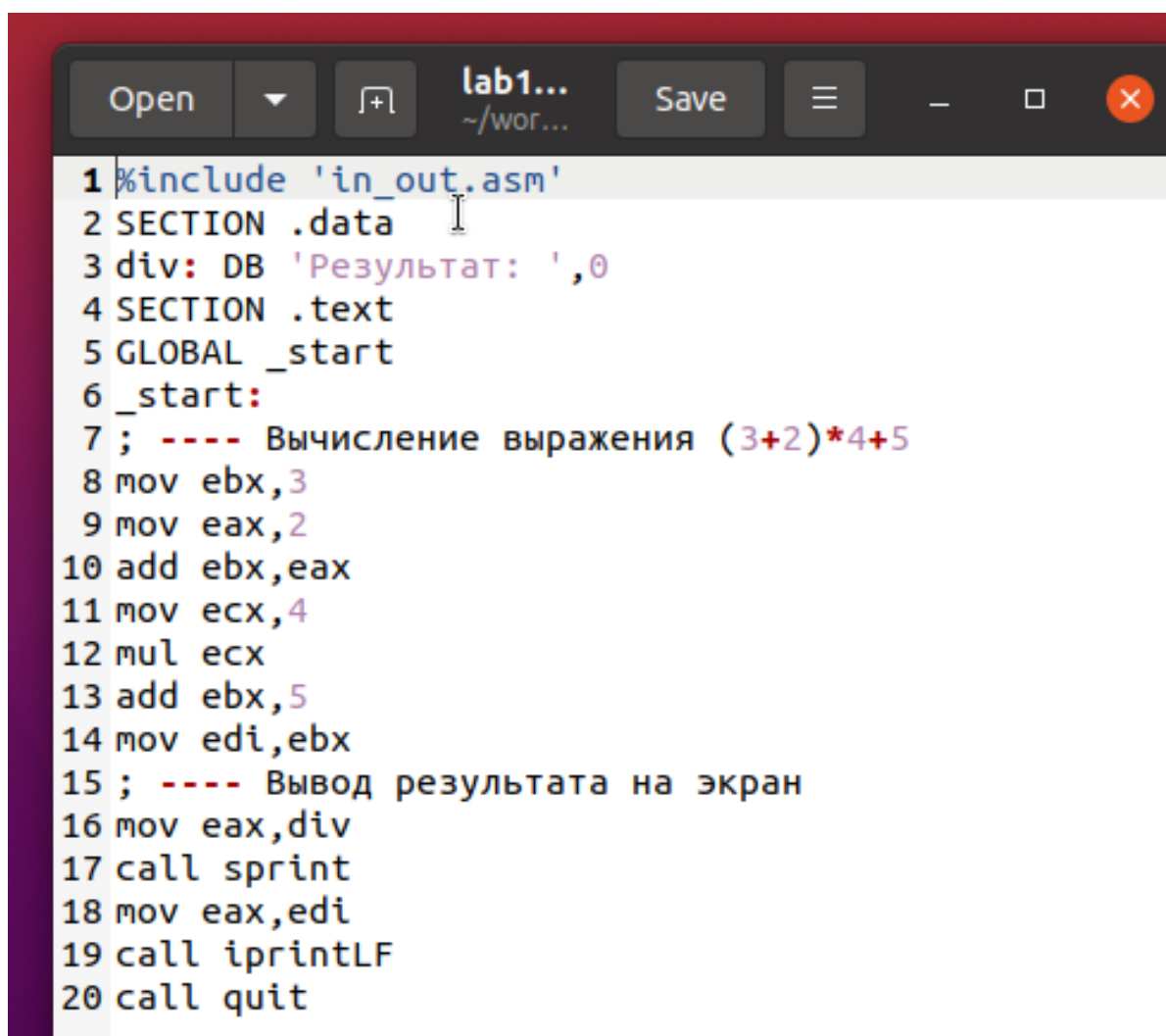
```

(gdb) q
avinash@avinash-VirtualBox:~/work/study/2022-2023/Архитектура
h-pc/labs/lab10$ nasm -f elf lab10-4.asm
avinash@avinash-VirtualBox:~/work/study/2022-2023/Архитектура
h-pc/labs/lab10$ ld -m elf_i386 -o lab10-4 lab10-4.o
avinash@avinash-VirtualBox:~/work/study/2022-2023/Архитектура
h-pc/labs/lab10$ ./lab10-4 1 2 3 4
f(x)=10(x-1)
Результат: 60
avinash@avinash-VirtualBox:~/work/study/2022-2023/Архитектура
h-pc/labs/lab10$

```

Рис. 2.17: Работа программы lab10-4.asm

7. В листинге 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. (рис. 2.18 2.19 2.20 2.21)



```
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
```

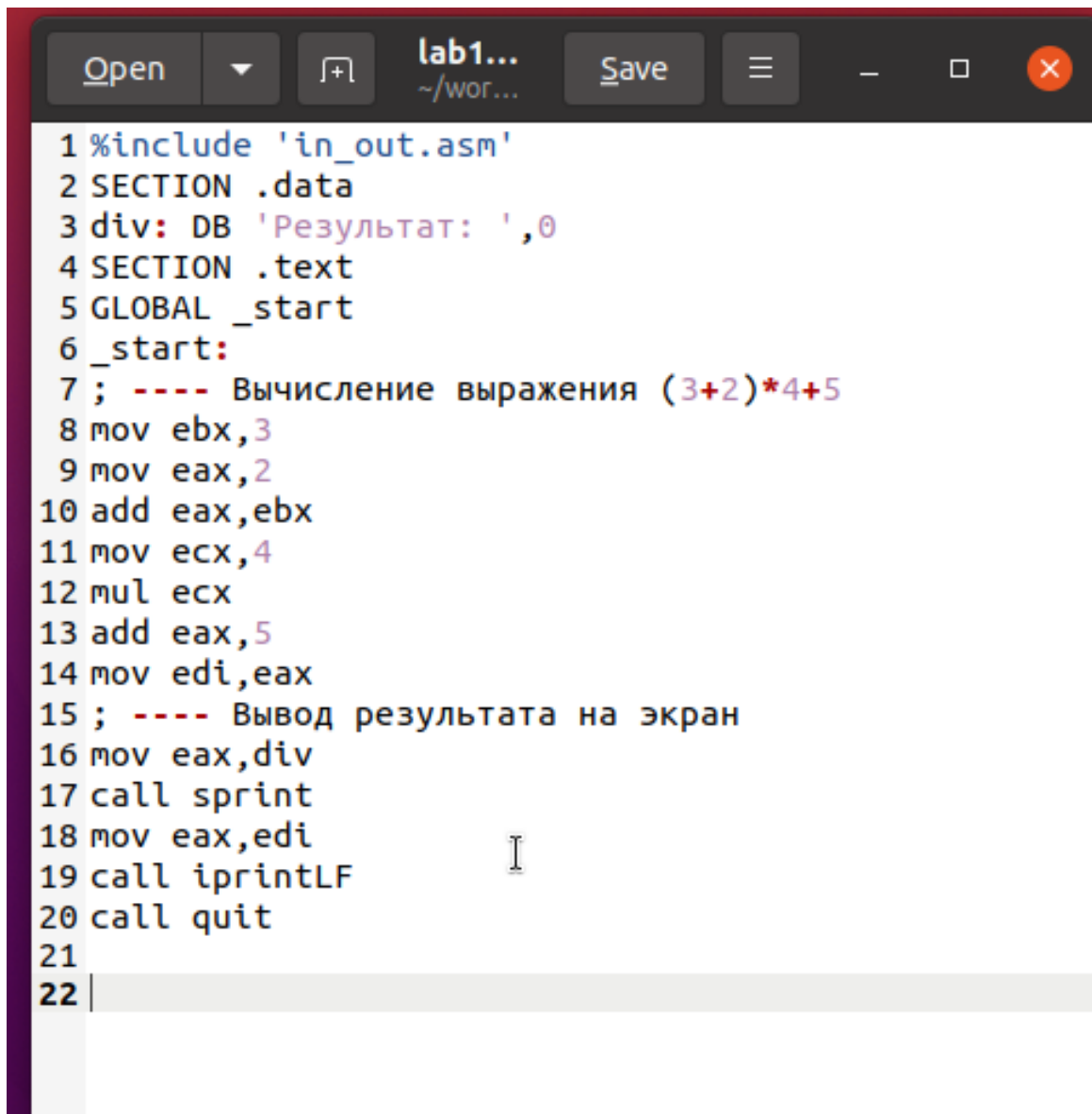
Рис. 2.18: код с ошибкой

```
avinish@avinash-VirtualBox: ~/work/study/2022-2023/Архитектура комп...
eax      0x804a000      134520832
eax      0x804a000      134520832
ecx      0x4           4
ebx      0x804a000      134520832
ebx      0xa          10
esp      0xffffd0d8     0xffffd0d8
esi      0x0           0
edi      0xa          10
edi      0xa          10
0x8049000 <slen>      push    ebx
0x804900f <sprint>     push    edx
>0x8049010 <sprint+1>  push    ecx      R [eax],0x0
0x8049011 <sprint+2>   push    ebx      d>
0x8049012 <sprint+3>   push    eax
0x8049013 <sprint+4>   call    0x8049000 <slen>har>
0x8049018 <sprint+9>   mov     edx,eax
0x804901a <sprint+11>  pop     eax
0x804901b <sprint+12>  mov     ecx,eax
native process 6653 In: nextchar      L??  PC: 0x8049003
(gdb) sprocess 6657 In: sprint        L??  PC: 0x8049010
(gdb) si
0x080490f2 in _start ()
(gdb) s
Single stepping until exit from function _start,
which has no line number information.
0x0804900f in sprint ()
(gdb) si
0x08049010 in sprint ()
(gdb) █
```

Рис. 2.19: отладка

Отметим, что перепутан порядок аргументов у инструкции add и что по окончании работы в edi отправляется ebx вместо eax





```
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 eax,ebx
11 mov ecx,4
12 mul ecx
13 add eax,5
14 mov edi,eax
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
21
22
```

Рис. 2.20: код исправлен

```
avinish@avinish-VirtualBox: ~/work/study/2022-2023/Архитектура комп...
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd0e0 0xffffd0e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

B+ 0x80490e8 <_start> mov ebx,0x3
B+ 0x80490e8 <_start>5> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
0x80490f2 <_start+10> add eax,ebx
0x80490f4 <_start+12> mov ecx,0x4
0x80490f9 <_start+17> mul ecx
0x80490fb <_start+19> add eax,0x5
0x80490fe <_start+22> mov edi,eax04a000
0x8049100 <_start+24> mov eax,0x804a000

native process 6667 In: _start L?? PC: 0x80490ed
StartinNo process In: h/work/study/2022-2023/ L?? PC: ??3

Breakpoint 1, 0x080490e8 in _start ()
(gdb) si
0x080490ed in _start ()
(gdb) cont
Continuing.
Результат: 25
[Inferior 1 (process 6667) exited normally]
(gdb) █
```

Рис. 2.21: проверка работы

## **3 Выводы**

Освоили работу с подпрограммами и отладчиком.