

# **Отчёта по лабораторной работе 10**

**Понятие подпрограммы. Отладчик GDB.**

Абрикосов Артем Камович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>5</b>	<b>Выводы</b>	<b>30</b>
	<b>Список литературы</b>	<b>31</b>

## Список иллюстраций

4.1	Файл lab10-1.asm . . . . .	9
4.2	Работа программы lab10-1.asm . . . . .	10
4.3	Файл lab10-1.asm . . . . .	11
4.4	Работа программы lab10-1.asm . . . . .	12
4.5	Файл lab10-2.asm . . . . .	13
4.6	Работа программы lab10-2.asm в отладчике . . . . .	14
4.7	дисассимилированный код . . . . .	15
4.8	дисассимилированный код в режиме интел . . . . .	16
4.9	точка остановки . . . . .	17
4.10	изменение регистров . . . . .	18
4.11	изменение регистров . . . . .	19
4.12	изменение значения переменной . . . . .	20
4.13	вывод значения регистра . . . . .	21
4.14	вывод значения регистра . . . . .	22
4.15	вывод значения регистра . . . . .	23
4.16	Файл lab10-4.asm . . . . .	24
4.17	Работа программы lab10-4.asm . . . . .	25
4.18	код с ошибкой . . . . .	26
4.19	отладка . . . . .	27
4.20	код исправлен . . . . .	28
4.21	проверка работы . . . . .	29

## **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 Задание

1. Изучите примеры реализации подпрограмм
2. Изучите работу с отладчиком GDB
3. Выполните самостоятельное задание
4. Загрузите файлы на GitHub.

### 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.

## 4 Выполнение лабораторной работы

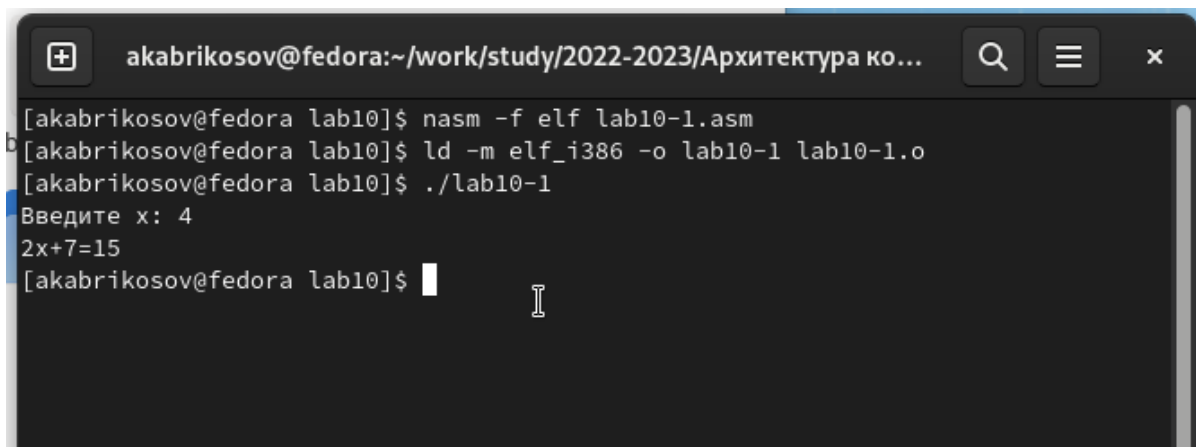
1. Создайте каталог для выполнения лабораторной работы № 10, перейдите в него и создайте файл lab10-1.asm:
2. В качестве примера рассмотрим программу вычисления арифметического выражения  $f(x) = 2x+7$  с помощью подпрограммы calcul. В данном примере  $x$  вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы (Листинг 10.1). (рис. 4.1, 4.2)



```
mc [akabrikosov@fedora]:~/work/study/2022-2023/Архитектура ..
lab10-1.asm [----] 0 L: [ 1+ 7 8/ 30] *(125 / 462b) 00
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
rez: 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, [rez]
call iprintLF
call quit
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [rez], eax
ret ; выход из подпрограммы
```

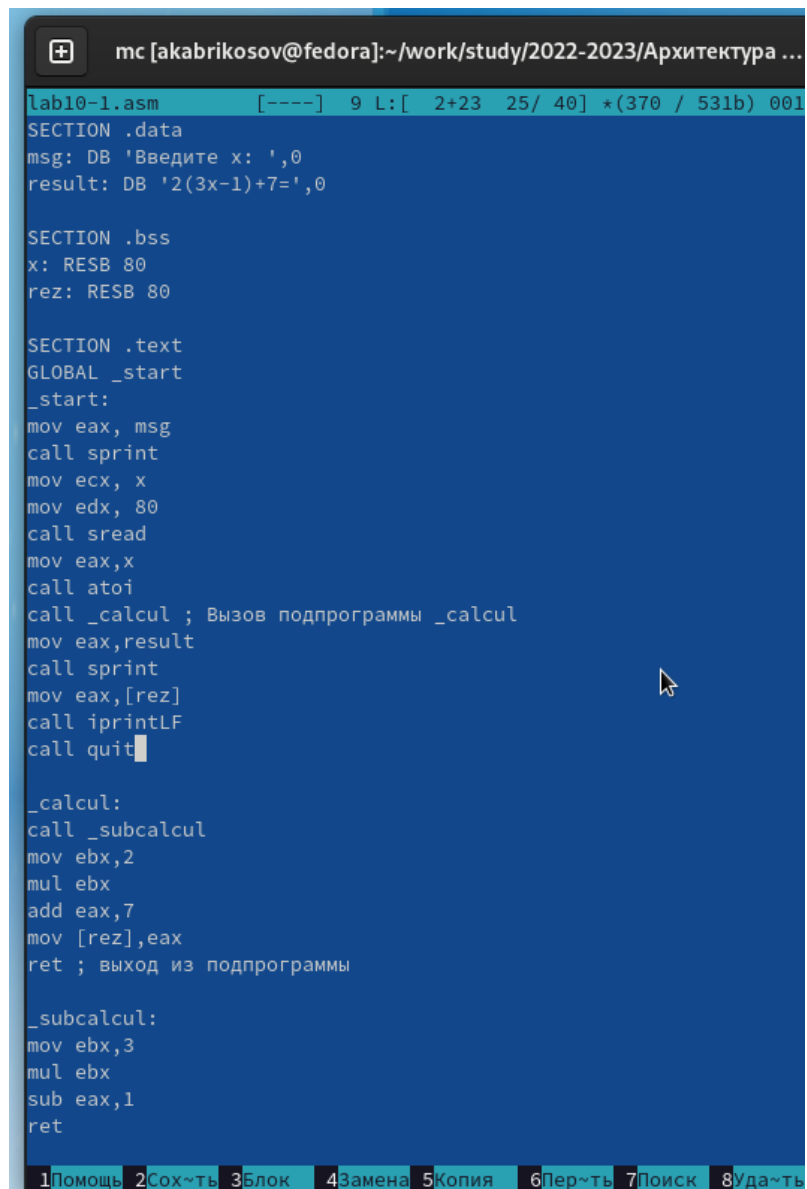
Рис. 4.1: Файл lab10-1.asm

A terminal window with a dark background and light text. The window title is "akabrikosov@fedora:~/work/study/2022-2023/Архитектура ко...". The terminal shows the following commands and output:

```
[akabrikosov@fedora lab10]$ nasm -f elf lab10-1.asm
[akabrikosov@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[akabrikosov@fedora lab10]$ ./lab10-1
Введите x: 4
2x+7=15
[akabrikosov@fedora lab10]$
```

Рис. 4.2: 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. 4.3, 4.4)



```
lab10-1.asm [----] 9 L:[ 2+23 25/ 40] *(370 / 531b) 001
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2(3x-1)+7=',0

SECTION .bss
x: RESB 80
rez: RESB 80

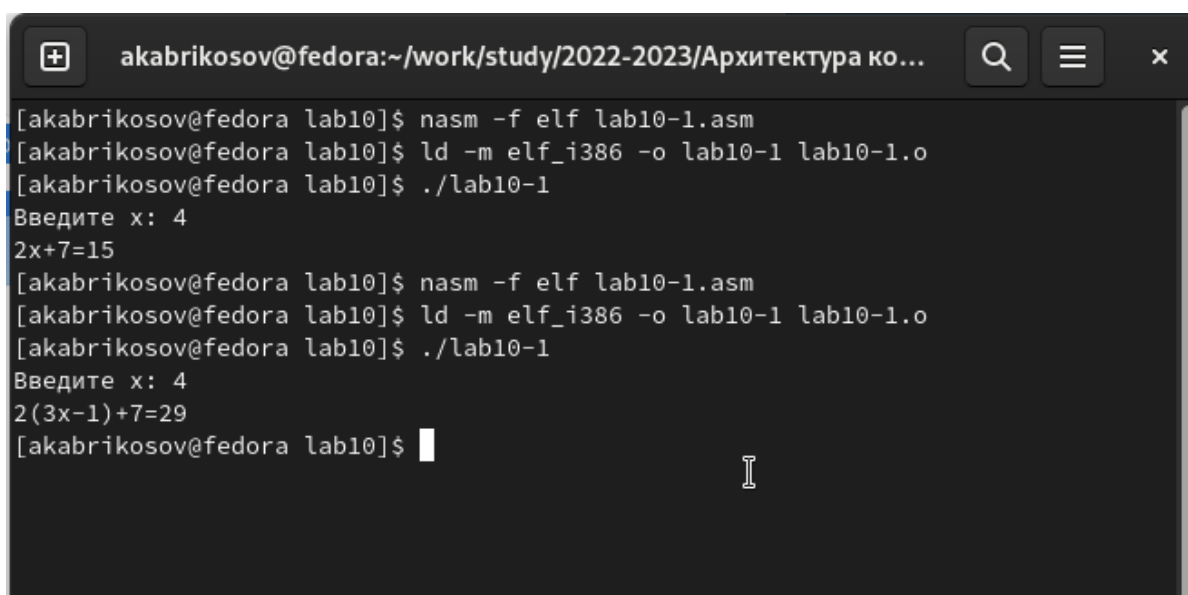
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [rez]
call iprintLF
call quit

_calcul:
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [rez], eax
ret ; выход из подпрограммы

_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret
```

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Выдать

Рис. 4.3: Файл lab10-1.asm



```
akabrikosov@fedora:~/work/study/2022-2023/Архитектура ко...
[akabrikosov@fedora lab10]$ nasm -f elf lab10-1.asm
[akabrikosov@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[akabrikosov@fedora lab10]$ ./lab10-1
Введите x: 4
2x+7=15
[akabrikosov@fedora lab10]$ nasm -f elf lab10-1.asm
[akabrikosov@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[akabrikosov@fedora lab10]$ ./lab10-1
Введите x: 4
2(3x-1)+7=29
[akabrikosov@fedora lab10]$
```

Рис. 4.4: Работа программы lab10-1.asm

4. Создайте файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!): (рис. 4.5)



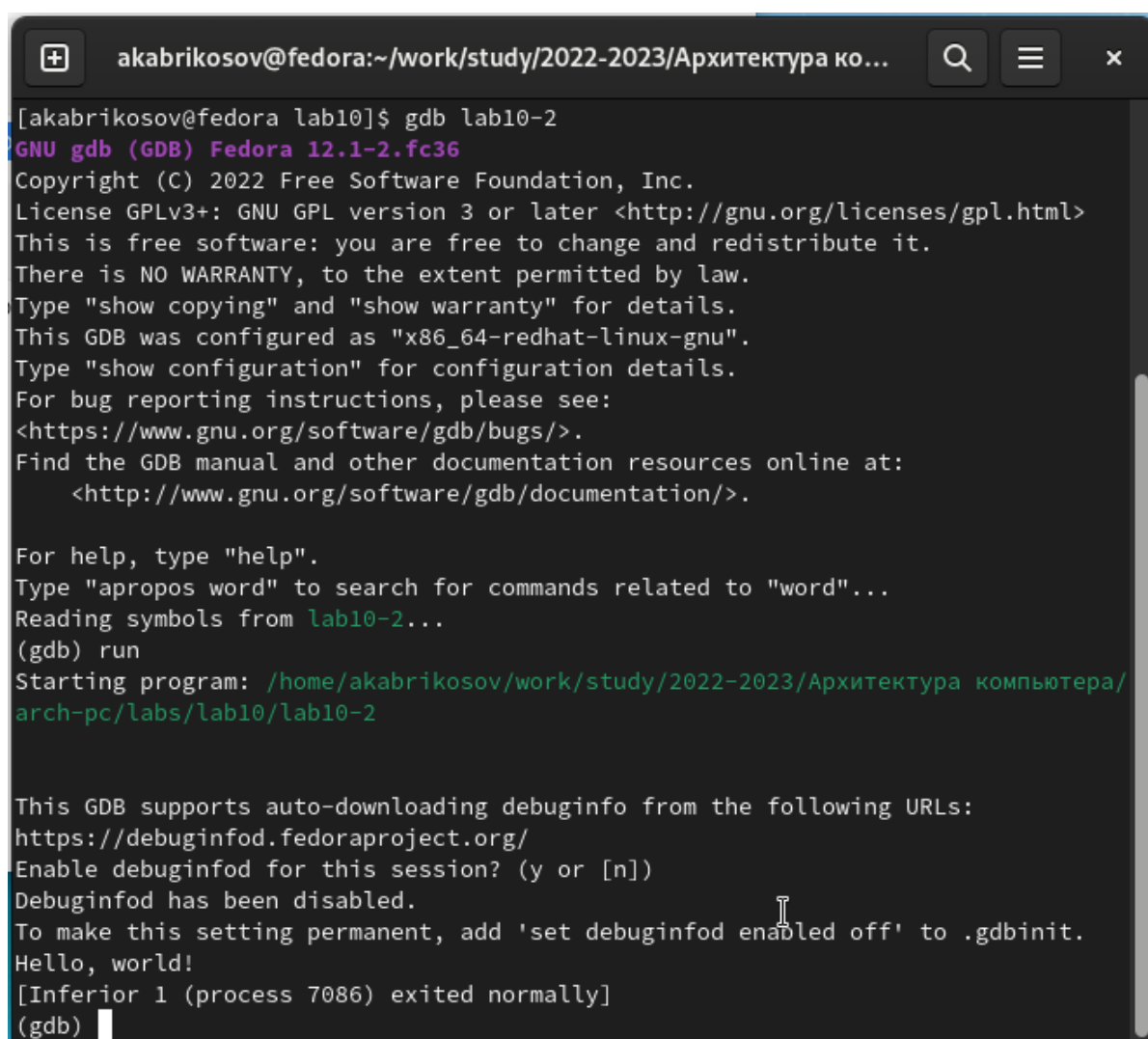
```
mc [akabrikosov@fedora]:~/work/study/2022-2
lab10-2.asm [----] 13 L:[ 1+17 18/ 24]
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2

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
```

Рис. 4.5: 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 команды run (сокращённо r):(рис. 4.6)

The image shows a terminal window with a dark background. The title bar at the top reads 'akabrikosov@fedora:~/work/study/2022-2023/Архитектура ко...'. The terminal content shows a GDB session starting with 'gdb lab10-2'. It displays the GNU GDB version '12.1-2.fc36' and copyright information for 2022. After the introductory text, the user enters 'run', and the program starts, displaying 'Hello, world!'. The session ends with '[Inferior 1 (process 7086) exited normally]' and the prompt '(gdb)'.

```
[akabrikosov@fedora lab10]$ gdb lab10-2
GNU gdb (GDB) Fedora 12.1-2.fc36
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-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 lab10-2...
(gdb) run
Starting program: /home/akabrikosov/work/study/2022-2023/Архитектура компьютера/
arch-pc/labs/lab10/lab10-2

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n])
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
Hello, world!
[Inferior 1 (process 7086) exited normally]
(gdb)
```

Рис. 4.6: 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. 4.7, 4.8)

```
akabrikosov@fedora:~/work/study/2022-2023/Архитектура ко...
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
Hello, world!
[Inferior 1 (process 7086) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 11.
(gdb) run
Starting program: /home/akabrikosov/work/study/2022-2023/Архитектура компьютера/
arch-pc/labs/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:11
11      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,%edx
      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. 4.7: дисассимилированный код

```
akabrikosov@fedora:~/work/study/2022-2023/Архитектура ко...
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) 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. 4.8: 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. 4.9)



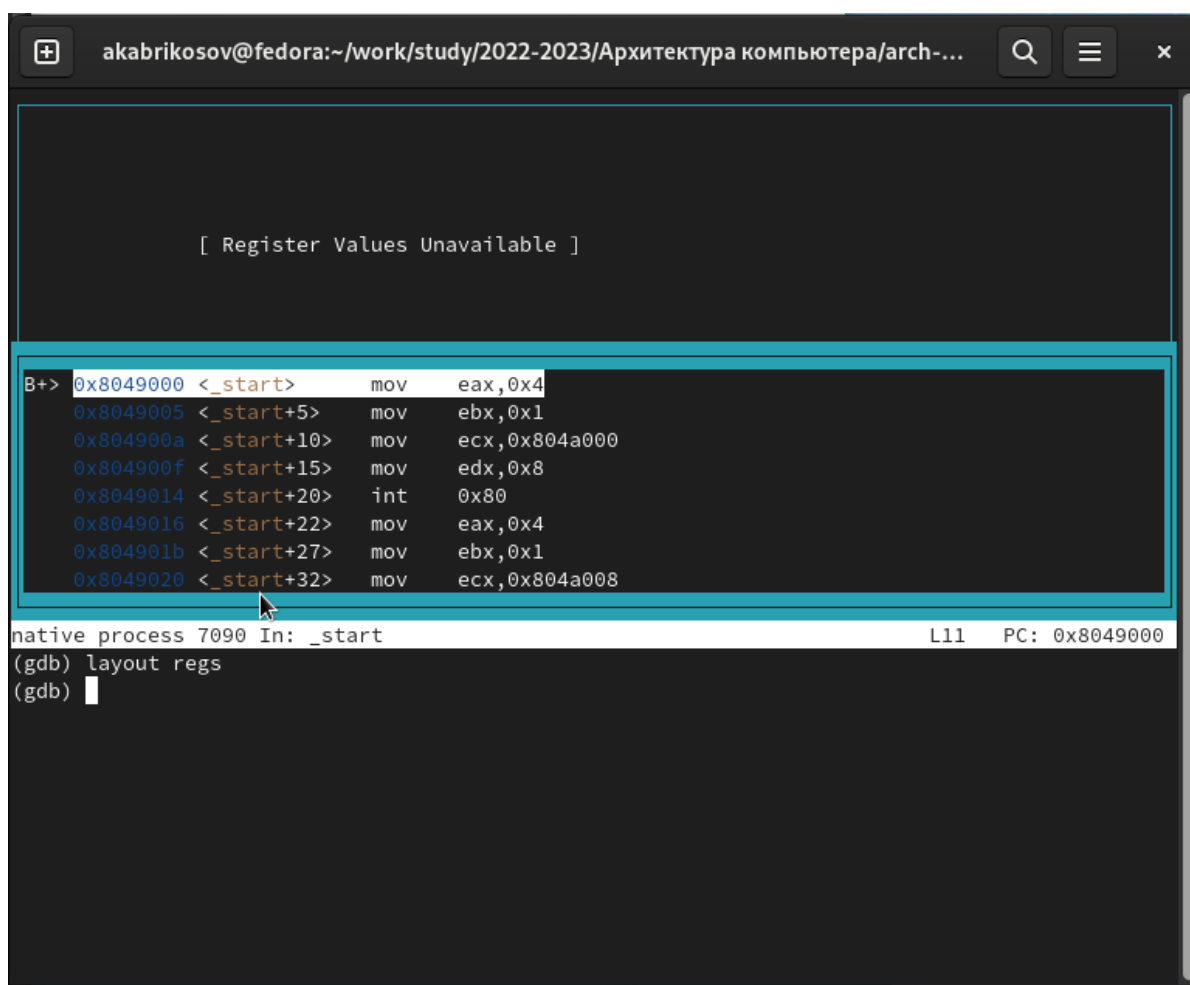


Рис. 4.9: точка остановки

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. 4.11 4.12)

```
akabrikosov@fedora:~/work/study/2022-2023/Архитектура компьютера/arch-...
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd140 0xffffd140
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 7090 In: _start L12 PC: 0x8049005
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--ss 0x2b
43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) si
(gdb)
```

Рис. 4.10: изменение регистров

```

akabrikosov@fedora:~/work/study/2022-2023/Архитектура компьютера/arch-...
eax      0x4      4
eax      0x1      1
edx      0x7      7
edx      0x7      7
esp      0xffffd140 0xffffd140
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

0x8049014 <_start+20> int 0x80
0x8049020 <_start+32> mov ecx,0x804a008
0x8049025 <_start+37> mov edx,0x7
> 0x804902a <_start+42> int 0x80
0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1
B+> 0x8049031 <_start+49> mov ebx,0x0
b+ 0x8049036 <_start+54> int 0x80
      8      add BYTE PTR [eax],al

native process 7090 In: _start L20 PC: 0x804902a
gs      0x0      0      2      31
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si

Breakpoint 2, _start () at lab10-2.asm:22
(gdb)

```

Рис. 4.11: 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. 4.12)

```
native process 7090 In: _start
gs          0x0          0
Breakpoint 2, _start () at lab10-2.asm:22
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}0x804a008='L'
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "Lor!d!\n\034"
(gdb) █
```

Рис. 4.12: 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. 4.13)

```
native process 7090 In: _start
```

```
$2 = 1
```

```
(gdb) p/s $ecx
```

```
$3 = 134520840
```

```
(gdb) p/x $ecx
```

```
$4 = 0x804a008
```

```
(gdb) p/s $edx
```

```
$5 = 7
```

```
(gdb) p/t $edx
```

```
$6 = 111
```

```
(gdb) p/x $edx
```

```
$7 = 0x7
```

```
(gdb) █
```

Рис. 4.13: вывод значения регистра

С помощью команды set измените значение регистра ebx:(рис. 4.14)

```
native process 7090 In: _start
$5 = 7
(gdb) p/t $edx
$6 = 111
(gdb) p/x $edx
$7 = 0x7
(gdb) set $ebx='2'
(gdb) p/s $ebx
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 2
(gdb) 
```

Рис. 4.14: вывод значения регистра

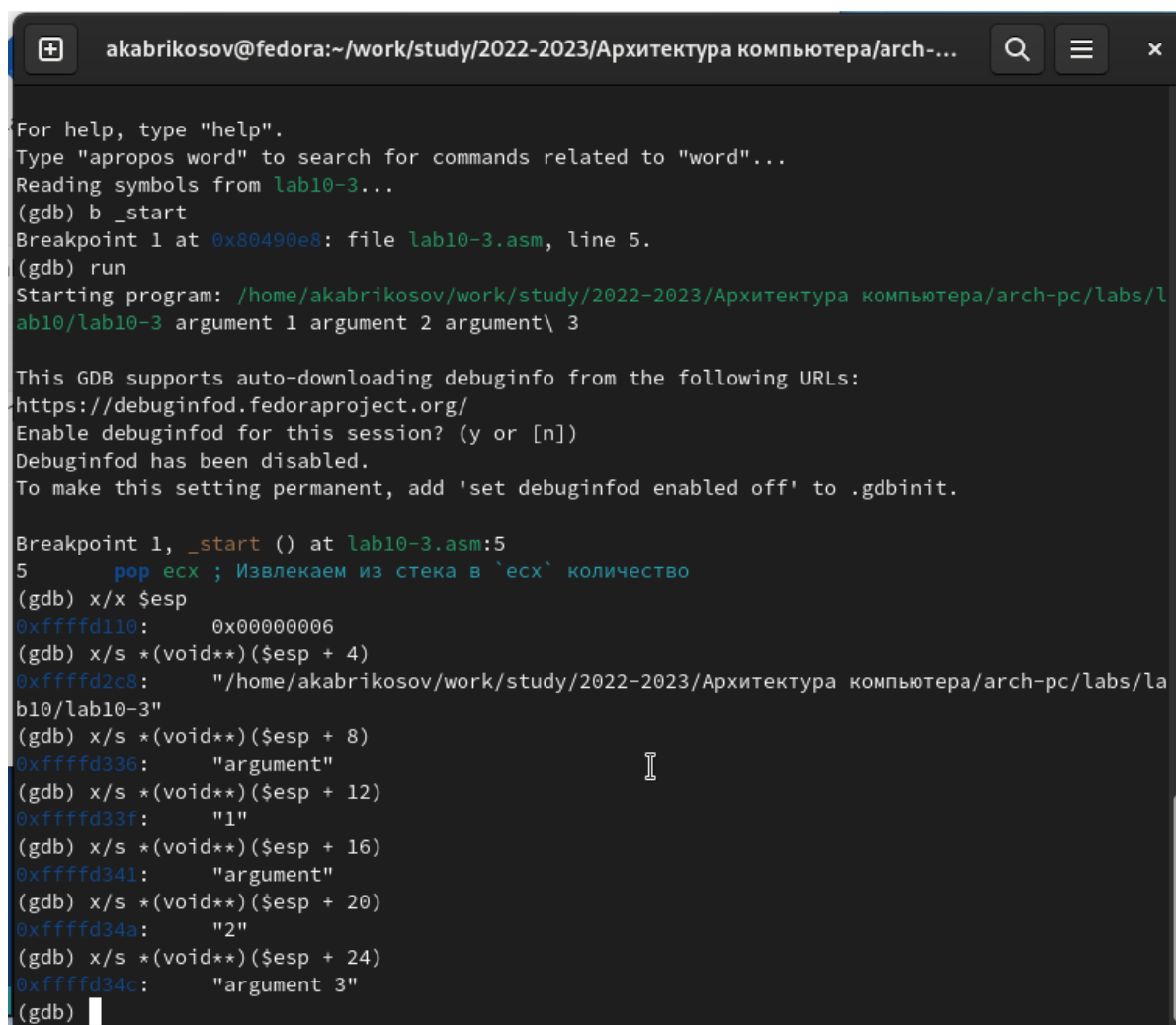
5. Скопируйте файл lab9-2.asm, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки. Создайте исполняемый файл. Для загрузки в gdb программы с аргументами необходимо использовать ключ `-args`. Загрузите исполняемый файл в отладчик, указав аргументы

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. 4.15)



```
akabrikosov@fedora:~/work/study/2022-2023/Архитектура компьютера/arch-...
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: file lab10-3.asm, line 5.
(gdb) run
Starting program: /home/akabrikosov/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10/lab10-3 argument 1 argument 2 argument\ 3

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

Breakpoint 1, _start () at lab10-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd110: 0x00000006
(gdb) x/s *(void**)(esp + 4)
0xffffd2c8: "/home/akabrikosov/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd336: "argument"
(gdb) x/s *(void**)(esp + 12)
0xffffd33f: "1"
(gdb) x/s *(void**)(esp + 16)
0xffffd341: "argument"
(gdb) x/s *(void**)(esp + 20)
0xffffd34a: "2"
(gdb) x/s *(void**)(esp + 24)
0xffffd34c: "argument 3"
(gdb)
```

Рис. 4.15: 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. 4.16 4.17)

```
mc [akabrikosov@fedora]:~/work/study/2022-2023/A
lab10-4.asm [----] 0 L: [ 1+24 25/ 38] *(27.
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
fx: db 'f(x)=7(x+1) ',0

SECTION .text
global _start
_start:
mov eax, fx
call sprintLF
pop ecx.
pop edx
sub ecx,1
mov esi, 0

next:
cmp ecx,0h
jz _end.
pop eax
call atoi
call calc
add esi,eax

loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

calc:
add eax,1
mov ebx,7.
mul ebx
ret
```

Рис. 4.16: Файл lab10-4.asm



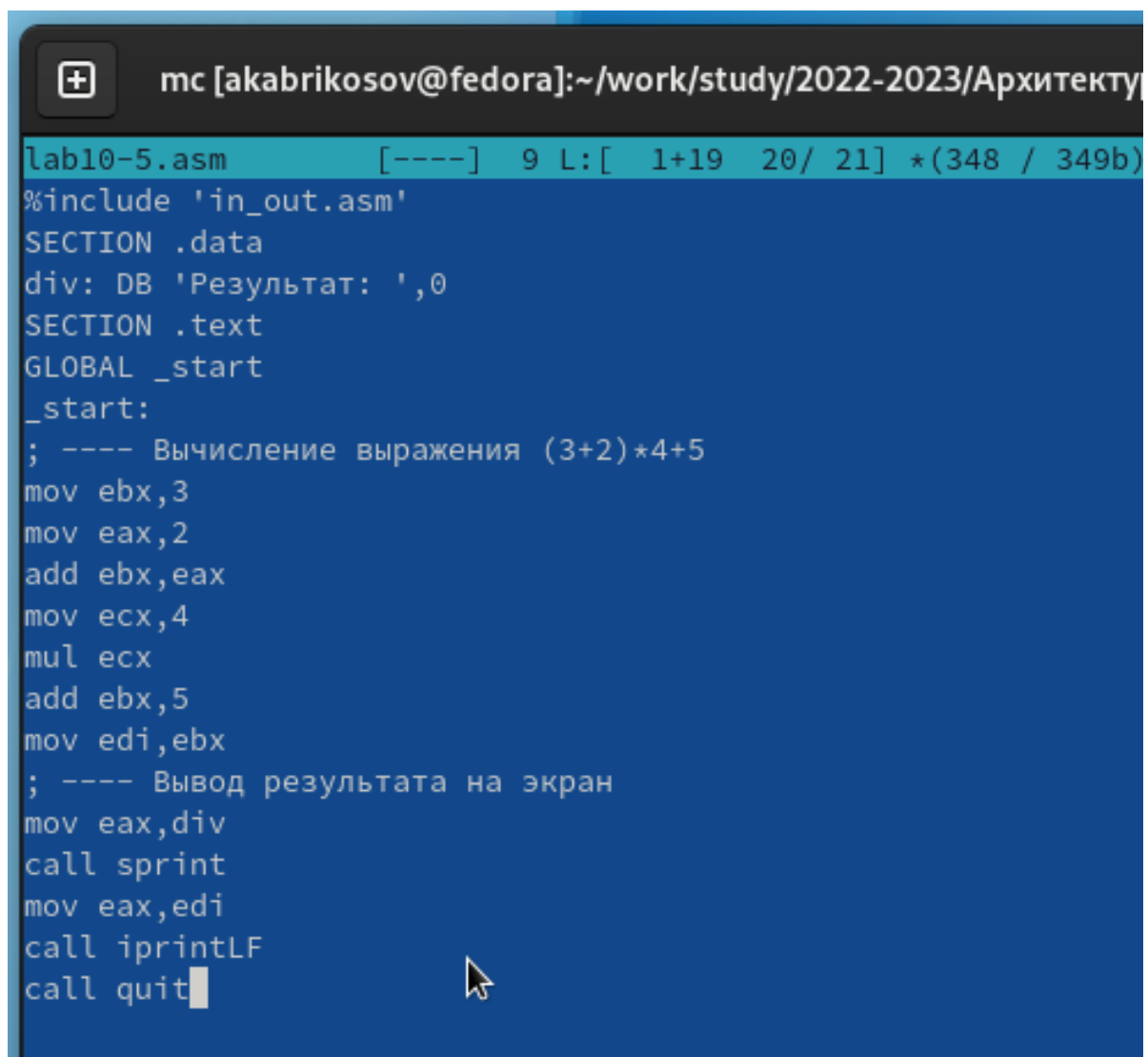
```

[akabrikosov@fedora lab10]$
[akabrikosov@fedora lab10]$
[akabrikosov@fedora lab10]$ nasm -f elf lab10-4.asm
[akabrikosov@fedora lab10]$ ld -m elf_i386 -o lab10-4 lab10-4.o
[akabrikosov@fedora lab10]$
[akabrikosov@fedora lab10]$ ./lab10-4 1 2 3
f(x)=7(x+1)
Результат: 63
[akabrikosov@fedora lab10]$ ./lab10-4 1 2 3 5 6 8 7 9 6 3 2
f(x)=7(x+1)
Результат: 441
[akabrikosov@fedora lab10]$

```

Рис. 4.17: 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. 4.18 4.19 4.20 4.21)



```
mc [akabrikosov@fedora]:~/work/study/2022-2023/Архитекту
lab10-5.asm [----] 9 L:[ 1+19 20/ 21] *(348 / 349b)
#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
```

Рис. 4.18: код с ошибкой

The screenshot shows a GDB debugger window with the following content:

akabrikosov@fedora:~/work/study/2022-2023/Архитектура компьютера/arch-...

Register group: general

Register	Value	Comment
eax	0x804a000	134520832
ecx	0x4	4
edx	0x0	0
ebx	0xa	10
esp	0xffffd140	0xffffd140
ebp	0x0	0x0
esi	0x0	0
edi	0xa	10
eip	0x8049105	0x8049105 <_start+29>

Assembly code:

```
0x80490f4 <_start+12> mov ecx,0x4
0x80490f9 <_start+17> mul ecx
0x80490fb <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
> 0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049086 <iprintLF>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add BYTE PTR [eax],al
```

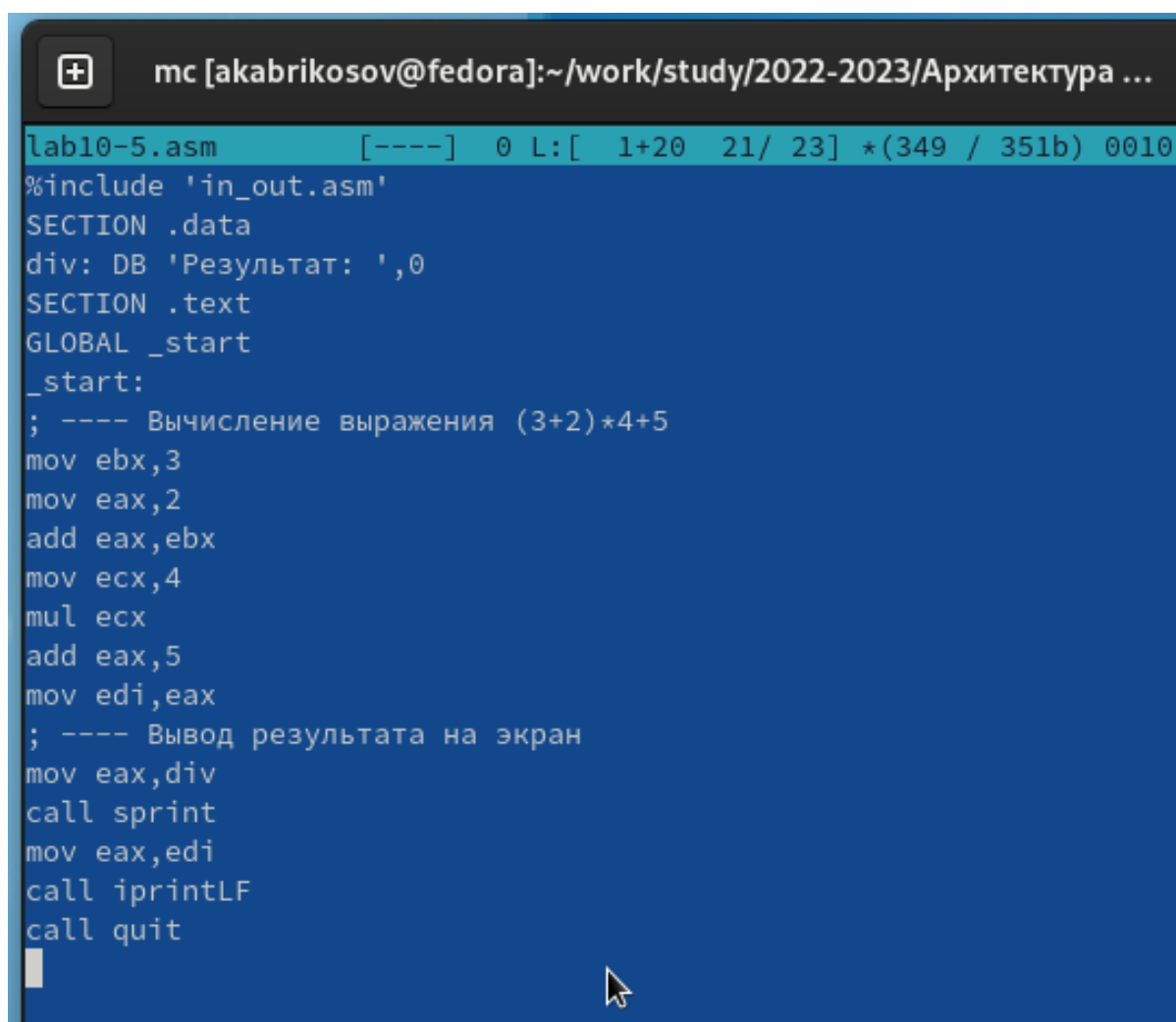
native process 7441 In: \_start L17 PC: 0x8049105

Breakpoint 1, \_start () at lab10-5.asm:8

```
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 4.19: отладка

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



```
mc [akabrikosov@fedora]:~/work/study/2022-2023/Архитектура ...
lab10-5.asm [----] 0 L:[ 1+20 21/ 23] *(349 / 351b) 0010
#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
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 4.20: код исправлен

```
akabrikosov@fedora:~/work/study/2022-2023/Архитектура компьютера/arch-...
eax      0x19      25
ecx      0x4       4
edx      0x0       0
ebx      0x3       3
esp      0xffffd140 0xffffd140
ebp      0x0       0x0
esi      0x0       0
edi      0x0       0
eip      0x80490fe 0x80490fe <_start+22>

B+ 0x80490e8 <_start>   mov     ebx,0x3
   0x80490fe <_start+22> mov     edi,eax
   0x8049100 <_start+24> add     eax,eb804a000
   0x8049105 <_start+29> call   0x804900f <sprint>
   0x804910a <_start+34> mul     eax,edi
   0x804910c <_start+36> call   0x8049086 <iprintLF>
> 0x8049111 <_start+41> call   0x80490db <quit>
                                04a000
                                rint>

native process 7539 In: _start L14 PC: 0x80490fe
No process In: L?? PC: ??
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) cont
Continuing.
Результат: 25
[Inferior 1 (process 7539) exited normally]
(gdb)
```

Рис. 4.21: проверка работы

## **5 Выводы**

Освоили работу с подпрограммами и отладчиком.

# Список литературы

1. Расширенный ассемблер: NASM
2. MASM, TASM, FASM, NASM под Windows и Linux