

# **Отчет по лабораторной работе №10**

**дисциплина: Архитектура компьютера**

Колобова Елизавета Андреевна гр. НММбд-01

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
2.1	Задание для самостоятельной работы . . . . .	18
<b>3</b>	<b>Выводы</b>	<b>23</b>

## Список иллюстраций

2.1	Рис. 1. Создание каталога для лаб. работы №10 и файла lab10-1.asm	6
2.2	Рис. 2. Ввод текста из листинга 10.1 . . . . .	7
2.3	Рис. 3. Компоновка и запуск файла lab10-1 . . . . .	7
2.4	Рис. 4. Изменение текста программы . . . . .	8
2.5	Рис. 5. Компоновка и запуск измененного файла . . . . .	9
2.6	Рис. 6. Ввод текста из листинга 10.2 . . . . .	9
2.7	Рис. 7. Компоновка файла и загрузка его в отладчик . . . . .	10
2.8	Рис. 8. Окно отладчика . . . . .	11
2.9	Рис. 9. Режим псевдографики . . . . .	12
2.10	Рис. 11. Просмотр информации о точке останова . . . . .	13
2.11	Рис. 12. Просмотр значения переменной . . . . .	14
2.12	Рис. 13. Замена символа и значения переменной . . . . .	15
2.13	Рис. 14. Изменение значения регистра ebx . . . . .	15
2.14	Рис. 15. Завершение выполнения программы . . . . .	16
2.15	Рис. 17. Создание исполняемого файла lab9-2 . . . . .	17
2.16	Рис. 18. Установка точки останова . . . . .	18
2.17	Рис. 19. Просмотр позиций стека . . . . .	18
2.18	Рис. 14. Текст программы . . . . .	19
2.19	Рис. 15. Компоновка и запуск файла . . . . .	19
2.20	Рис. 14. Текст программы . . . . .	21
2.21	Рис. 15. Отладка программы . . . . .	22
2.22	Рис. 16. Отладка программы . . . . .	22

## **Список таблиц**

# 1 Цель работы

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

## 2 Выполнение лабораторной работы

1. Создадим каталог для программ лабораторной работы No 10, перейдем в него и создадим файл lab10-1.asm (рис. 2.1):

```
mkdir ~/work/arch-pc/lab10
```

```
cd ~/work/arch-pc/lab10
```

```
touch lab10-1.asm
```

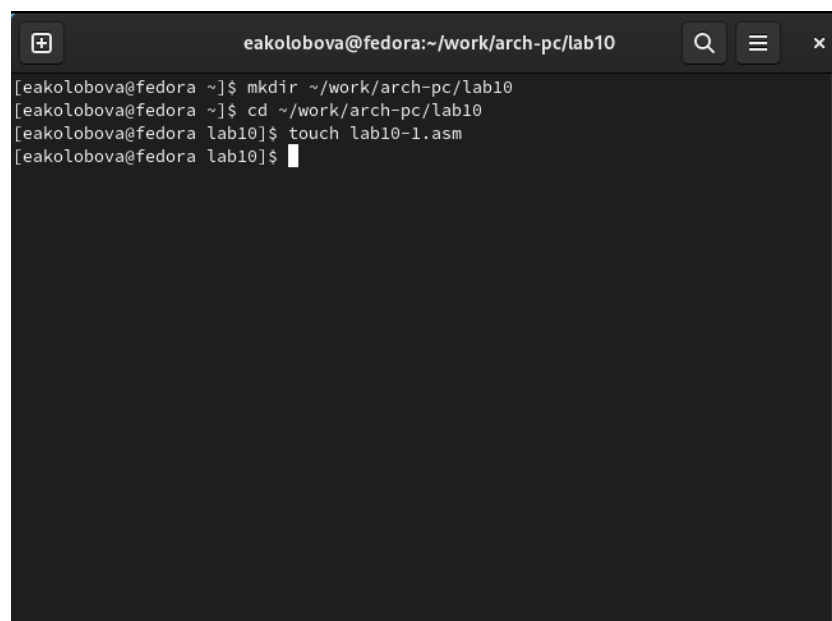
A screenshot of a terminal window with a dark background. The window title is 'eakolobova@fedora:~/work/arch-pc/lab10'. The terminal shows the following commands and their outputs: '[eakolobova@fedora ~]\$ mkdir ~/work/arch-pc/lab10', '[eakolobova@fedora ~]\$ cd ~/work/arch-pc/lab10', '[eakolobova@fedora lab10]\$ touch lab10-1.asm', and '[eakolobova@fedora lab10]\$' with a cursor at the end. The window has standard Linux window controls (minimize, maximize, close) and search, menu, and close buttons in the title bar.

Рис. 2.1: Рис. 1. Создание каталога для лаб. работы №10 и файла lab10-1.asm

2. 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. Введем в файл

lab10-1.asm текст программы из листинга 10.1. Создадим исполняемый файл и проверим его работу (рис. 2.2), рис. 2.3).

```

lab10-1.asm  [----]  3 L: [ 1+37 38/ 38] *(386 / 386b) <EOF>  [*][X]
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=' ,0

SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi

call _calcul

mov eax, result
call sprint
mov eax, [res]
call iprintLF

call quit

_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret
  
```

Рис. 2.2: Рис. 2. Ввод текста из листинга 10.1

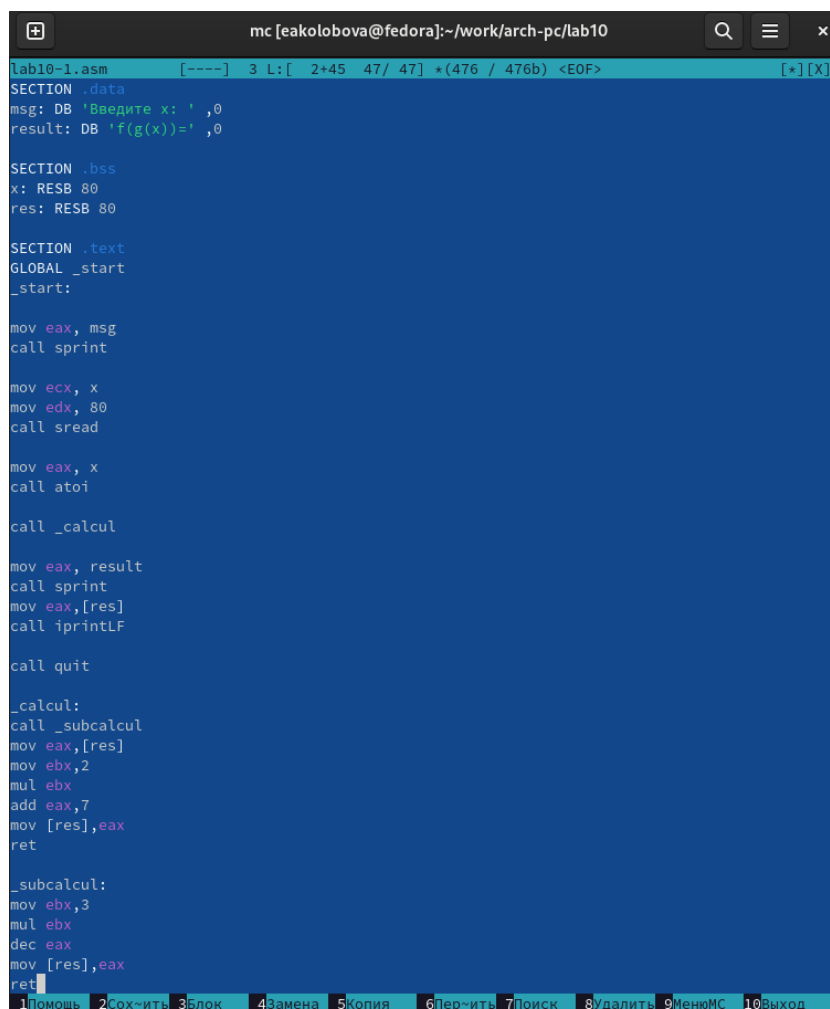
```

[eakolobova@fedora lab10]$ nasm -f elf -g lab10-1.asm
[eakolobova@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[eakolobova@fedora lab10]$ ./lab10-1
Введите x: 4
2x+7=15
[eakolobova@fedora lab10]$ ./lab10-1
Введите x: 8
2x+7=23
[eakolobova@fedora lab10]$
  
```

Рис. 2.3: Рис. 3. Компоновка и запуск файла lab10-1

3. Изменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиату-

ры,  $f(x)=2x+7$ ,  $g(x)=3x-1$ . Т.е.  $x$  передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение  $g(x)$ , результат возвращается в `_calcul` и вычисляется выражение  $f(g(x))$ . Результат возвращается в основную программу для вывода результата на экран. (рис. 2.4, 2.5):



```
mc [eakolobova@fedora]:~/work/arch-pc/lab10
lab10-1.asm  [----]  3 L: [  2+45  47/ 47] *(476 / 476b) <EOF>  [*] [X]
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x))=',0

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

mov eax, result
call sprint
mov eax, [res]
call iprintfLF

call quit

_calcul:
call _subcalcul
mov eax, [res]
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret

_subcalcul:
mov ebx, 3
mul ebx
dec eax
mov [res], eax
ret
```

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Удалить 9МенюМС 10Выход

Рис. 2.4: Рис. 4. Изменение текста программы



```

[eakolobova@fedora lab10]$ nasm -f elf -g lab10-1.asm
[eakolobova@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[eakolobova@fedora lab10]$ ./lab10-1
Введите x: 2
f(g(x))=17
[eakolobova@fedora lab10]$ ./lab10-1
Введите x: 5
f(g(x))=35
[eakolobova@fedora lab10]$

```

Рис. 2.5: Рис. 5. Компоновка и запуск измененного файла

4. Создадим файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!) (рис. 2.6)

```

lab10-2.asm  [-M--]  0 L: [ 1+ 8  9/ 26] *(133 / 299b) 0010 0x00A  [*] [X]
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

```

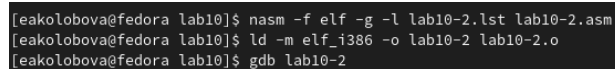
Рис. 2.6: Рис. 6. Ввод текста из листинга 10.2

Получим исполняемый файл. Для работы с GDB трансляцию программ необходимо проводить с ключом '-g'. (рис. 2.7)

```
nasm -f elf -g -l lab10-2.lst lab10-2.asm
ld -m elf_i386 -o lab10-2 lab10-2.o
```

Загрузим исполняемый файл в отладчик gdb (рис. 2.7):

```
user@dk4n31:~$ gdb lab10-2
```



```
[eakolobova@fedora lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[eakolobova@fedora lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
[eakolobova@fedora lab10]$ gdb lab10-2
```

Рис. 2.7: Рис. 7. Компоновка файла и загрузка его в отладчик

Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 2.8):

```
(gdb) run
Starting program: ~/work/arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 10220) exited normally]
(gdb)
```

Для более подробного анализа программы установим брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустим её. (рис. 2.8)

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 12.
(gdb) run
Starting program: ~/work/arch-pc/lab10/lab10-2
Breakpoint 1, _start () at lab10-2.asm:12
12 mov eax, 4
```



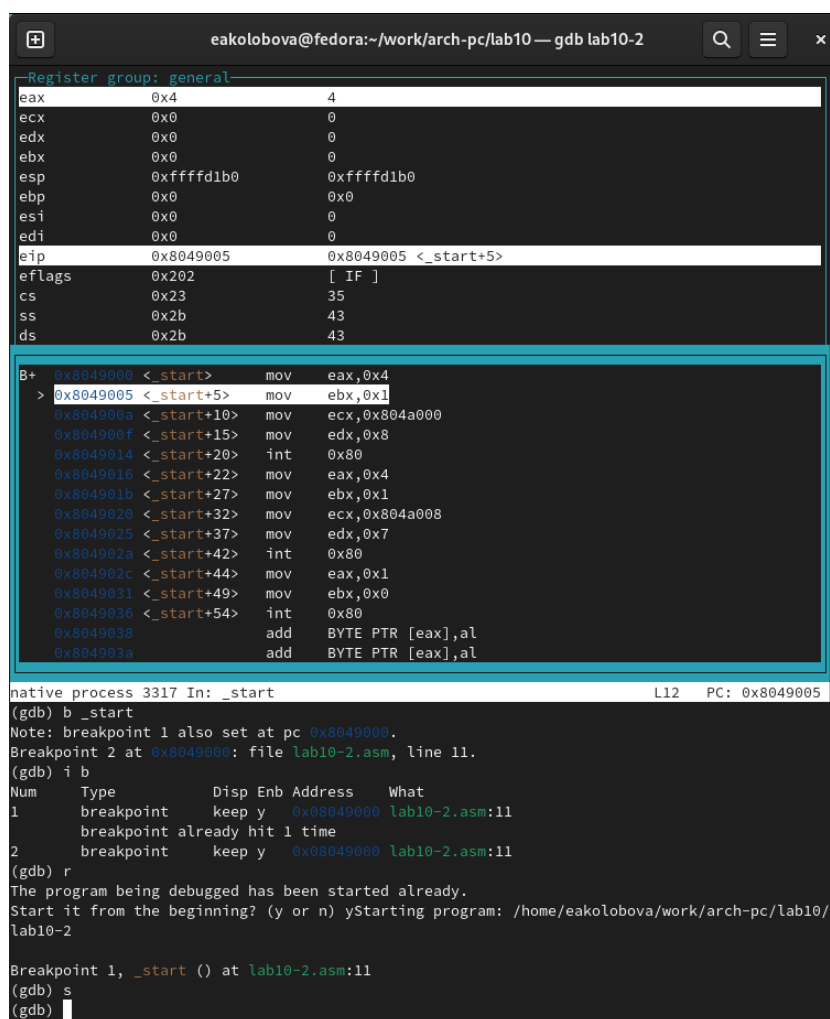
Перечислите различия отображения синтаксиса машинных команд в режимах АТТ и Intel.

5. Включим режим псевдографики для более удобного анализа программы (рис. 2.9):

```
(gdb) layout asm
```

```
(gdb) layout regs
```

Установим еще одну точку останова по адресу инструкции. Определим адрес предпоследней инструкции (`mov ebx,0x0`) и установим точку останова. (рис. 2.9)



```
eakolobova@fedora:~/work/arch-pc/lab10 — gdb lab10-2
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43

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
0x8049038 add BYTE PTR [eax],al
0x804903a add BYTE PTR [eax],al

native process 3317 In: _start L12 PC: 0x8049005
(gdb) b _start
Note: breakpoint 1 also set at pc 0x8049000.
Breakpoint 2 at 0x8049000: file lab10-2.asm, line 11.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab10-2.asm:11
breakpoint already hit 1 time
2 breakpoint keep y 0x08049000 lab10-2.asm:11
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/eakolobova/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:11
(gdb) s
(gdb)
```

Рис. 2.9: Рис. 9. Режим псевдографики

Посмотрим информацию о всех установленных точках останова (рис. 2.10):

(gdb) i b

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/eakolobova/work/arch-pc/lab10/
lab10-2

Breakpoint 1, _start () at lab10-2.asm:11
(gdb) s
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab10-2.asm:11
          breakpoint already hit 1 time
2        breakpoint      keep y   0x08049000 lab10-2.asm:11
          breakpoint already hit 1 time
(gdb)
```

Рис. 2.10: Рис. 11. Просмотр информации о точке останова

Посмотрим значение переменной `msg1` по имени (рис. 2.11)

(gdb) x/1sb &msg1

0x804a000 <msg1>: "Hello, "

```

eakolobova@fedora:~/work/arch-pc/lab10 — gdb lab10-2
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43

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
b+ 0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80
0x8049038 add BYTE PTR [eax],al
0x804903a add BYTE PTR [eax],al

native process 3317 In: _start L12 PC: 0x8049005
ebx      0x0      0
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
--Type <RET> for more, q to quit, c to continue without paging--qQuit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)

```

Рис. 2.11: Рис. 12. Просмотр значения переменной

Посмотрим значение переменной `msg2` по адресу. Посмотрим инструкцию `mov ecx,msg2` которая записывает в регистр `ecx` адрес переменной `msg`. Изменим первый символ переменной `msg1` (рис. 2.12):

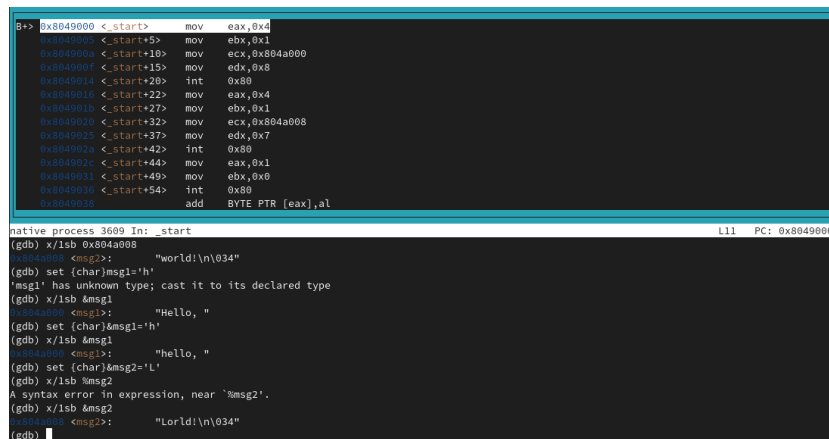
```

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

```

Заменим символ во второй переменной `msg2`. Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде)

значение регистра edx. (рис. 2.12)



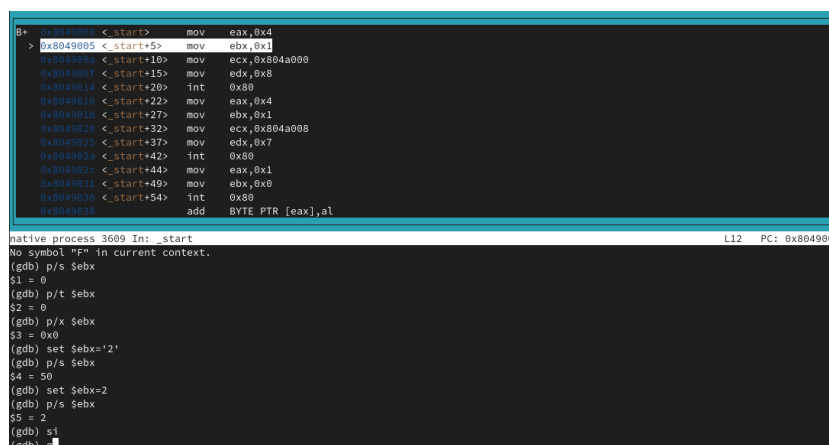
```
B> 0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a008
0x804900f <_start+15> mov edx,0x8
0x8049014 <_start+20> int 0x80
0x8049019 <_start+25> mov eax,0x4
0x804901e <_start+30> mov ebx,0x1
0x8049023 <_start+35> mov ecx,0x804a008
0x8049028 <_start+40> mov edx,0x7
0x804902d <_start+45> int 0x80
0x8049032 <_start+50> mov eax,0x1
0x8049037 <_start+55> mov ebx,0x0
0x804903c <_start+60> int 0x80
0x8049041 <_start+65> add BYTE PTR [eax],al

native process 3609 In: _start L11 PC: 0x8049000
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb) set (char)msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) x/1sb &msg1
0x804a00c <msg1>: "Hello, "
(gdb) set (char)&msg1='h'
(gdb) x/1sb &msg1
0x804a00c <msg1>: "hello, "
(gdb) set (char)&msg2='L'
(gdb) x/1sb &msg2
A syntax error in expression, near '&msg2'.
(gdb) x/1sb &msg2
0x804a010 <msg2>: "Lor!\n\034"
(gdb)
```

Рис. 2.12: Рис. 13. Замена символа и значения переменной

С помощью команды set изменим значение регистра ebx (рис. 2.13):

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$3 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$4 = 2
(gdb)
```



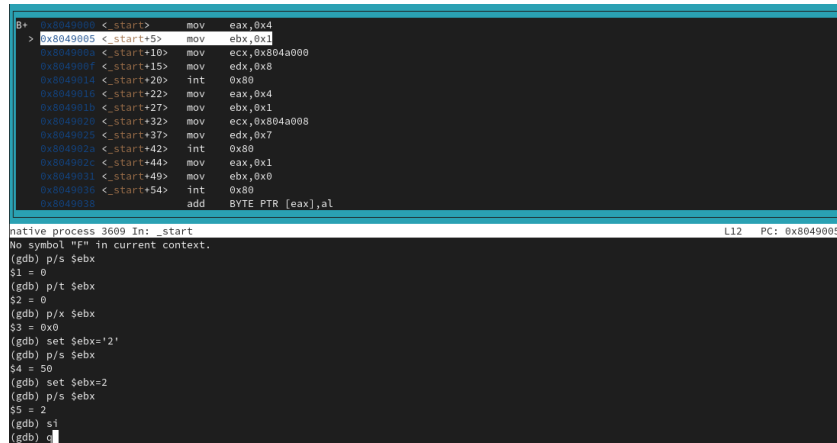
```
B> 0x8049000 <_start> mov eax,0x4
> 0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a008
0x804900f <_start+15> mov edx,0x8
0x8049014 <_start+20> int 0x80
0x8049019 <_start+25> mov eax,0x4
0x804901e <_start+30> mov ebx,0x1
0x8049023 <_start+35> mov ecx,0x804a008
0x8049028 <_start+40> mov edx,0x7
0x804902d <_start+45> int 0x80
0x8049032 <_start+50> mov eax,0x1
0x8049037 <_start+55> mov ebx,0x0
0x804903c <_start+60> int 0x80
0x8049041 <_start+65> add BYTE PTR [eax],al

native process 3609 In: _start L12 PC: 0x8049005
No symbol "F" in current context.
(gdb) p/s $ebx
$1 = 0
(gdb) p/t $ebx
$2 = 0
(gdb) p/x $ebx
$3 = 0x0
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb) s1
(gdb)
```

Рис. 2.13: Рис. 14. Изменение значения регистра ebx

Объясните разницу вывода команд `p/s $ebx`. Эта команда выводит строку, оканчивающуюся нулем.

Завершим выполнение программы с помощью команды `continue` (сокращенно `c`) и выйдем из GDB с помощью команды `quit` (сокращенно `q`). (рис. 2.14)



```
0x8049000 <_start> mov eax,0x4
> 0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a008
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
0x8049038 add BYTE PTR [eax],al

native process 3609 In: _start
No symbol "F" in current context.
(gdb) p/s $ebx
$1 = 0
(gdb) p/t $ebx
$2 = 0
(gdb) p/x $ebx
$3 = 0x0
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb) s1
(gdb) q
```

Рис. 2.14: Рис. 15. Завершение выполнения программы

6. Скопируем файл `lab9-2.asm`, созданный при выполнении лабораторной работы No9, с программой выводящей на экран аргументы командной строки (Листинг 9.2) в файл с именем `lab10-3.asm` (рис. 2.15):

```
cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
```

Создадим исполняемый файл. (рис. 2.15)

```
nasm -f elf -g -l lab10-3.lst lab10-3.asm
ld -m elf_i386 -o lab10-3 lab10-3.o
```

Загрузим исполняемый файл в отладчик, указав аргументы (рис. 2.15):

```
gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
```



```

0> 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
0x804903c      add     BYTE PTR [eax],al
native process 3609 In: _start
No symbol "F" in current context.
(gdb) p/s $ebx
$1 = 0
(gdb) p/t $ebx
$2 = 0
(gdb) p/x $ebx
$3 = 0x0
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb) s1
(gdb) c

```

Рис. 2.15: Рис. 17. Создание исполняемого файла lab9-2

Для начала установим точку останова перед первой инструкцией в программе и запустим ее. (рис. 2.16)

```
(gdb) b _start
(gdb) run
```

Адрес вершины стека хранится в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы):

```
(gdb) x/x $esp
0xffffd200: 0x05
```

Как видно, число аргументов равно 5 – это имя программы lab10-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Посмотрим остальные позиции стека (рис. 2.17)

Почему шаг изменения адреса равен 4, т.к. программа запускается с четырьмя (вместе с названием программы) аргументами

```

(gdb) b _start
Breakpoint 1 at 0x80490e3: file lab10-3.asm, line 7.
(gdb) run
Starting program: /home/eakolobova/work/arch-pc/lab10/lab10-3 1 2 6

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

Breakpoint 1, _start () at lab10-3.asm:7
7      pop ecx
(gdb) x/x $esp
Value can't be converted to integer.
(gdb)

```

Рис. 2.16: Рис. 18. Установка точки останова

```

eakolobova@fedora:~/work/arch-pc/lab10 — gdb --args lab10-3 аргумент1 аргумент2 аргумент3
(gdb) layout asm
[eakolobova@fedora lab10]$ gdb --args lab10-3 аргумент1 аргумент2 'аргумент3'
GNU gdb (GDB) Fedora 11.2-3.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 lab10-3...
(gdb) b _start
Breakpoint 1 at 0x80490e3: file lab10-3.asm, line 7.
(gdb) run
Starting program: /home/eakolobova/work/arch-pc/lab10/lab10-3 аргумент1 аргумент2 аргумент3

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

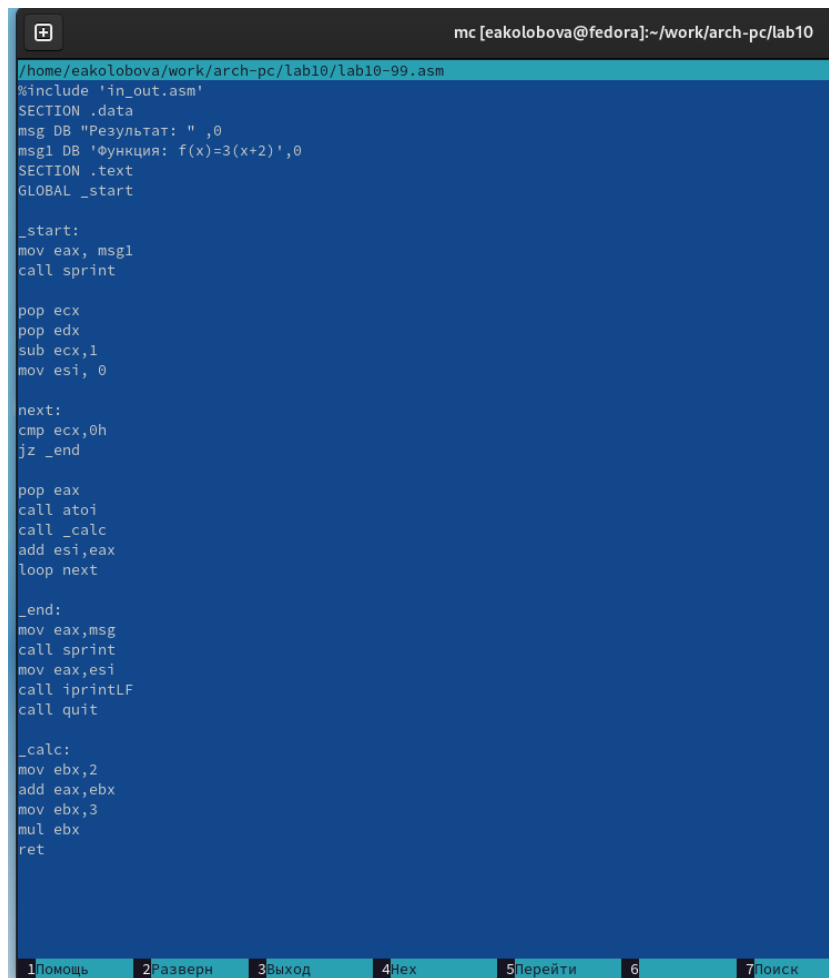
Breakpoint 1, _start () at lab10-3.asm:7
7      pop ecx
(gdb) x/x $esp
0xffffd100: 0x00000004
(gdb) x/s *(void**)(esp + 4)
0xffffd332: "/home/eakolobova/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd33c: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd344: "аргумент2"
(gdb) x/s *(void**)(esp + 16)
0xffffd34c: "аргумент3"
(gdb) x/s *(void**)(esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb) x/s *(void**)(esp + 24)
0xffffd354: "SHELL=/bin/bash"
(gdb)

```

Рис. 2.17: Рис. 19. Просмотр позиций стека

## 2.1 Задание для самостоятельной работы

1. Преобразовать программу из лабораторной работы No9 (Задание No1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$ , как подпрограмму. (рис. 2.18, 2.19)



```
mc [eakolobova@fedora]:~/work/arch-pc/lab10
/home/eakolobova/work/arch-pc/lab10/lab10-99.asm
%include 'in_out.asm'
SECTION .data
msg DB "Результат: " ,0
msg1 DB 'Функция: f(x)=3(x+2)',0
SECTION .text
GLOBAL _start
_start:
mov eax, msg1
call sprint

pop ecx
pop edx
sub ecx,1
mov esi, 0

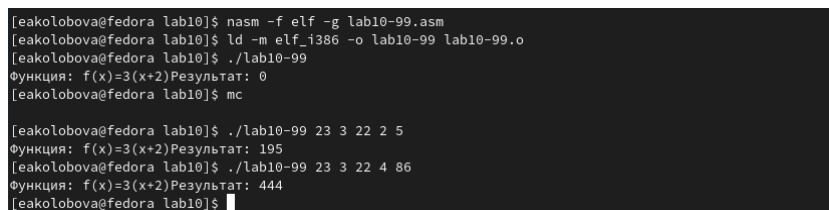
next:
cmp ecx,0h
jz _end

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:
mov ebx,2
add eax,ebx
mov ebx,3
mul ebx
ret
```

Рис. 2.18: Рис. 14. Текст программы



```
[eakolobova@fedora lab10]$ nasm -f elf -g lab10-99.asm
[eakolobova@fedora lab10]$ ld -m elf_i386 -o lab10-99 lab10-99.o
[eakolobova@fedora lab10]$ ./lab10-99
Функция: f(x)=3(x+2)Результат: 0
[eakolobova@fedora lab10]$ mc

[eakolobova@fedora lab10]$ ./lab10-99 23 3 22 2 5
Функция: f(x)=3(x+2)Результат: 195
[eakolobova@fedora lab10]$ ./lab10-99 23 3 22 4 86
Функция: f(x)=3(x+2)Результат: 444
[eakolobova@fedora lab10]$
```

Рис. 2.19: Рис. 15. Компоновка и запуск файла

- В листинге 10.3 приведена программа вычисления выражения  $(3+2)*4+5$ . При запуске данная программа дает неверный результат. Проверить это. С помощью отладчика GDB, анализируя изменения значений регистров, определить ошибку и исправить ее.

Как видно на рис. 2.21, 2.22, результат сложения  $3+2$  записывается в регистр `ebx`, а перемножаются после этого значения регистров `ecx` и `eax` вместо `ecx` и `ebx`, как предполагается в тексте программы. После этого к значению регистра `ebx` - 5 - прибавляется 5, и этот результат программа выводит, как конечное значение. Ошибка заключается в том, что для операции умножения суммы на значение `ecx`, первый множитель - сумма - находится в регистре `ebx`, а умножается всегда регистр `eax`. (рис. 2.20, 2.21, 2.22)



```
lab10-91.asm [----] 20 L:[ 1+ 2 3/
#include 'in_our.asm'
SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:

mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx

mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 2.20: Рис. 14. Текст программы

```

eakolobova@fedora:~/work/arch-pc/lab10 — gdb lab10-91

Register group: general
eax 0x2 2 ecx 0x4 4
edx 0x0 0 ebx 0x5 5
esp 0xffffd190 0xffffd190 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
ebp 0x80490f9 0x80490f9 <_start+17> eflags 0x206 [ PF IF ]
cs 0x23 35Register Values Unavailable ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

b- 0x80490e9 <_start> mov ebx,0x3
0x80490ed <_start+5> cmp Bax,0x2R [eax],0x0
0x80490f7 <_start+10> add ebx,eax d>
0x80490f9 <_start+12> mov ecx,0x4
B> 0x80490f9 <_start+17> mul ecx,0x003 <nextchar>
0x80490fb <_start+19> add ebx,0x5
0x80490fd <_start+21> mov edi
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x804900e <printf>
0x8049111 <_start+41> call 0x804900b <quit>
0x8049116 add BYTE PTR [eax],al
0x8049119 add BYTE PTR [eax],al

exec No process in:
native process 6115 In: _start L7? PC: 7?
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/eakolobova/work/arch-pc/lab10/lab10-91

Breakpoint 1, _start () at lab10-91.asm:9
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) nProgram not restarted.
(gdb) disable breakpoint 1
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/eakolobova/work/arch-pc/lab10/lab10-91

Breakpoint 2, _start () at lab10-91.asm:13
(gdb)

```

Рис. 2.21: Рис. 15. Отладка программы

```

eakolobova@fedora:~/work/arch-pc/lab10 — gdb lab10-91

Register group: general
eax 0x8 8 ecx 0x4 4
edx 0x0 0 ebx 0xa 10
esp 0xffffd190 0xffffd190 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
ebp 0x80490fe 0x80490fe <_start+22> eflags 0x206 [ PF IF ]
cs 0x23 35Register Values Unavailable ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

b- 0x80490f9 <_start+17> mul ecx,0x003 <nextchar>
0x80490fb <_start+19> add ebx,0x5
B> 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 0x804900e <printf>
0x8049111 <_start+41> call 0x804900b <quit>
0x8049116 add BYTE PTR [eax],al
0x8049119 add BYTE PTR [eax],al
0x804911a add BYTE PTR [eax],al
0x804911b add BYTE PTR [eax],al
0x804911c add BYTE PTR [eax],al
0x804911d add BYTE PTR [eax],al

exec No process in:
native process 6121 In: _start L15 PC: 0x80490fe
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/eakolobova/work/arch-pc/lab10/lab10-91

Breakpoint 2, _start () at lab10-91.asm:13
(gdb) b 0x80490fe
Function "0x80490fe" not defined.
(gdb) b *0x80490fe
Breakpoint 3 at 0x80490fe: file lab10-91.asm, line 15.
(gdb) disable breakpoint 2
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/eakolobova/work/arch-pc/lab10/lab10-91

Breakpoint 3, _start () at lab10-91.asm:15
(gdb)

```

Рис. 2.22: Рис. 16. Отладка программы

Ссылка на репозиторий: [https://github.com/eakolobova/study\\_2022-2023\\_arch-pc/tree/master/labs/lab010/report](https://github.com/eakolobova/study_2022-2023_arch-pc/tree/master/labs/lab010/report)

## 3 Выводы

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